jamk.fi

Mobiililaitteen ja palvelimen välinen salaus lähiverkossa

Juho Halttunen

Opinnäytetyö
Toukokuu 2017
Tekniikan ja liikenteen ala
Insinööri (AMK), Tietotekniikan koulutusohjelma

Jyväskylän ammattikorkeakoulu
JAMK University of Applied Sciences

jamk.fi

Kuvailulehti

Tekijä(t)	Julkaisun laji	Päivämäärä
Halttunen, Juho	Opinnäytetyö, AMK	Toukokuu 2017
	Sivumäärä 47 + 2	Julkaisun kieli Suomi
		Verkkojulkaisulupa myönnetty: x

Työn nimi

Mobiiliapplikaation ja palvelimen välinen salaus lähiverkossa

Tutkinto-ohjelma

Tietotekniikan koulutusohjelma

Työn ohjaaja(t)

Juha Jokinen, Antti Häkkinen

Toimeksiantaja(t)

Cozify Oy, Kimmo Ruotoistenmäki

Tiivistelmä

Opinnäytetyön toimeksiantajana toimi Cozify Oy. Cozify valmistaa kotiautomaatiolaitetta jonka avulla voidaan ohjata eri valmistajien langattomia laitteita.

Tavoitteena oli toteuttaa mobiiliapplikaation ja palvelimen välille luotettava salaus lähiverkossa. Salauksen lisäksi mobiiliapplikaation ja palvelimen välinen tunnistautuminen tuli olla luotettavaa. Salauksen täytyy vastata nykypäivän turvallisuusvaatimuksia sekä tulevaisuuden haasteita.

Opinnäytetyössä tutkittiin mahdollisia ratkaisuja toteuttaa salaus mahdollisimman yksinkertaisesti sekä soveltuvuutta hybridimobiiliapplikaatioon. Hybridimobiiliapplikaatiolla on olemassa omat rajoitteensa, minkä vuoksi tutkiminen keskittyi SSL-salauksen luomiseen.

Opinnäytetyön lopputuloksena löydettiin, mahdollisuus toteuttaa salaus SSL-protokollalla käyttämällä sertifikaattia. Salauksella on kuitenkin omat rajoituksensa toteutuksessa. Toimeksiantajan on tämän työn pohjalta mahdollisuus lähteä kehittämään omaa ympäristöään, jolla salaus voidaan luoda.

Avainsanat (asiasanat)

Cordova, HTTPS, Mobiiliapplikaatio, Openssl, Palvelin, Salaus,

Muut tiedot

jamk.fi

Description

Author(s)	Type of publication	Date
Halttunen, Juho	Bachelor's thesis	May 2017
		Language of publication: Finnish
	Number of pages 47 + 2	Permission for web publication: x

Title of publication

Traffic encryption between mobile application and server

Degree programme

Information Technology

Supervisor(s)

Jokinen Juha, Häkkinen Antti

Assigned by

Cozify Oy, Kimmo Ruotoistenmäki

Abstract

The bachelor's thesis was assigned by Cozify Oy a company that manufactures home automation equipment. With that equipment, Iot devices from different manufacturers can be controlled.

The goal of this thesis was to create encryption between mobile application and server in LAN network. In addition, there was need for authentication between those devices. Encryption should meet the security today's requirements and future challenges.

First, encryption protocols were analyzed studied to find out which of them would be simple and applicable to the hybrid mobile application. Because there are some limitations due the hybrid mobile application, the thesis was concentrated to research SSL protocol.

As a result of the thesis, it was discovered that it is possible to create encryption between hybrid mobile application and server with SSL protocol. The encryption can be implemented with certificates; however, there are some limitations with this approach.

In the future, Cozify Oy can develop their own environment to create the needed encryption based on this thesis.

Keywords/tags (subjects)

Cordova, Encryption, HTTPS, Mobile application, Openssl

Miscellaneous

Sisältö

Kuviot

Lyhenteet

CA	Certificate Authority
CN	Common Name
DHCP	Dynamic Host Configuration Protocol
FQDN	Fully Qualified Domain Name
HTTP	Hypertext Transfer Protocol
HTTPS	Hypertext Transfer Protocol Secure
IETF	Internet Engineering Task Force
IoT	Internet of Things
IP	Internet Protocol
LAN	Local Area Network
IMEI	International Mobile Equipment Identity
MitM	Man in the Middle
OSI	Open Systems Interconnection
SSH	Secure Shell
SSL	Secure Sockets Layer
SSL VPN	Secure Sockets Layer Virtual Private Network
TLS	Transport Layer Security
VPN	Virtual Private Network

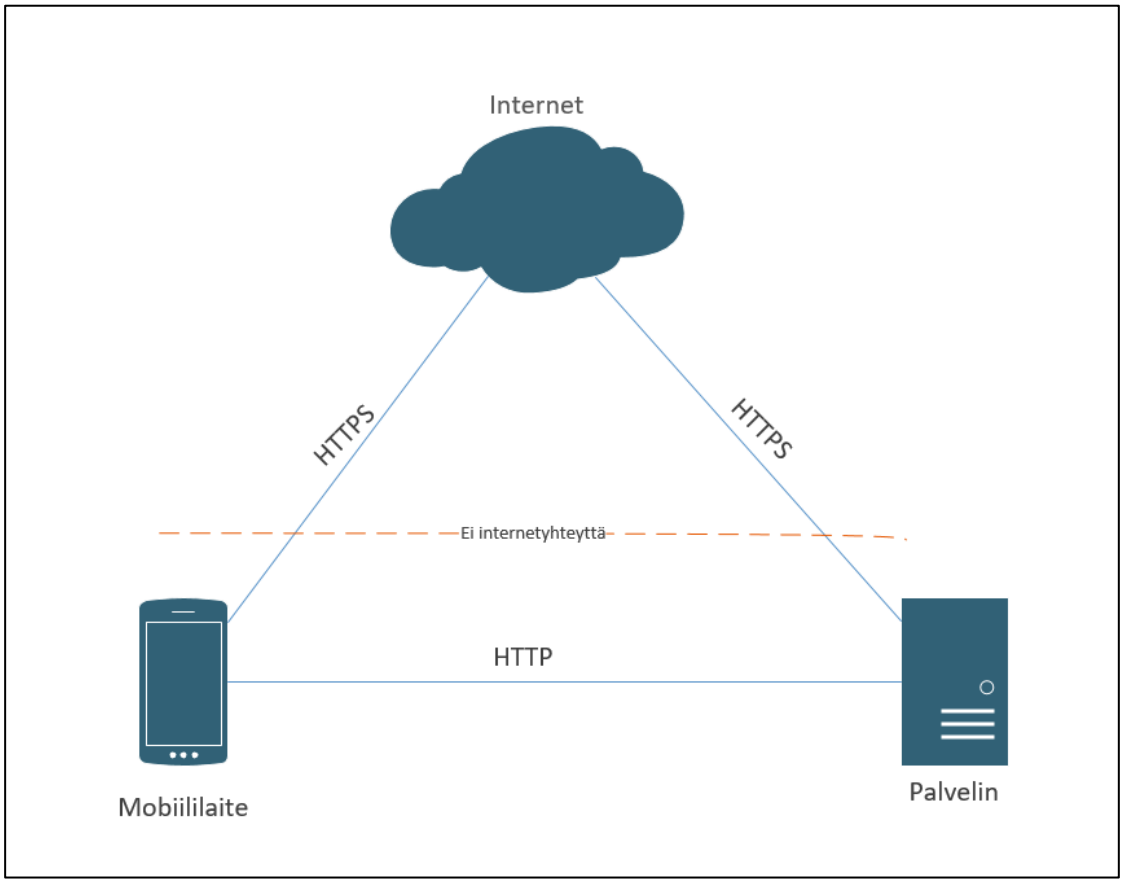
1 Työn lähtökohdat

1.1 Toimeksiantaja

Opinnäytetyön toimeksiantaja oli Cozify Oy. Se on suomalainen yritys, joka tuottaa kotiautomaatiolaitteita. Näistä kotiautomaatiolaitteista käytetään nimitystä Cozify Hub. Niiden avulla voidaan yhdistää eri valmistajien langattomia älylaitteita eli niin kutsuttuja Iot-laitteita yhdeksi kokonaisuudeksi. Hubia voidaan ohjata suoraan mobiililaitteella Cozify-sovelluksella. (Cozify Oy 2017.)

1.2 Tavoitteet

Opinnäytetyön tavoitteena oli tutkia ja toteuttaa mahdollinen salaus mobiililaitteen ja palvelimen välille lähiverkossa. Laitteiden tulisi pystyä tunnistamaan toisensa luotettavasti ilman internetyhteyttä. Kuviossa 1 on havainnollistettu kuvalla opinnäytetyön lähtökohtaa. Siinä on kuvattu, kuinka mobiililaite muodostaa suoran yhteyden lähiverkossa HTTP:n avulla palvelimen kanssa, mikäli internetyhteyttä ei muodosteta. Internetyhteyden ollessa toiminnassa mobiililaite muodostaa internetin kautta palvelimeen HTTPS-liikenteen. Tämä tapahtuu pilven kautta, minne sekä mobiililaite että palvelin ottavat yhteyden. Tällöin liikenne kierrätetään pilven kautta, jolloin liikenne voidaan salata HTTPS-protokollalla.



Kuvio 1. Opinnäytetyön lähtökohta

Verkkoliikenteen salauksen lisäksi tulisi olla mahdollista, että mobiililaite sekä palvelin tunnistaisivat toisensa luotettavasti. Tämän avulla saataisiin lisättyä tietoturvaa merkittävästi. Muu huomion arvoinen asia on käytetty mobiilisovellus. Se on niin kutsuttu hybrid-applikaatio ja se luo omat lähtökohtansa työhön. On otettava huomioon hybrid-applikaation luomat rajoitteet mitä tulee käytettäviin ratkaisumalleihin. Tämä tarkoittaa, ettei kaikkia mahdollisia verkkoliikenteen salaukseen tarkoitettuja protokollia voida välttämättä käyttää tai nämä eivät välttämättä ole tarpeeksi yksinkertaisesti implementoitavissa mobiiliapplikaatioon.

Hybrid-applikaatiolle on olemassa salauksen muodostamiseen ratkaisu. Tämä ratkaisumalli kuitenkin vaatii mobiililaitteelle erityisiä kommunikointikirjastoja. Tämänkaltaiset ratkaisut eivät ole mielekkäitä, sillä ne on implementoitava jokaiselle mobiilialustalle erikseen, mikä taas kasvattaa applikaation kehittäjän työnmäärää. Lisäksi ratkaisun tulisi pystyä hyödyntämään hybridsovelluksen tapaa kommunikoida selaimen avulla.

1.3 Ajankohtaisuus

Tällä hetkellä kiinostus IoT-laitteita kohtaan on kovassa nousussa niin Suomessa kuin muualla maailmalla. Yhä nopeammin kasvava IoT on myös mahdollinen turvallisuusriski tavallisille käyttäjille, sillä niitä hankkiessa ei yleensä mietitä kotiverkon turvallisuutta. Mahdollisten Iot-laitteissa olevien tietoturva-aukkojen myötä hyökkääjä voi päästä käsiksi kodin verkkoon. Ulkopuolisen päästessä kotiverkkoon sisälle voi hänelle avautua mahdollisuus hallita kodin muita laitteita, jotka sijaitsevat samassa kotiverkossa. Tällä tavalla voi syntyä erittäin suuria tietoturvariskejä.

2 Salausmenetelmät

Tietoliikennettä voidaan salata monella eri tavalla verkossa. Liikenteen salaamiseen on olemassa lukemattomia keinoja, joista yleisin tapa salata verkkoliikenne selaimen ja palvelimen välillä on käyttää HTTPS-salausta.

2.1 HTTPS

2.1.1 Yleistä

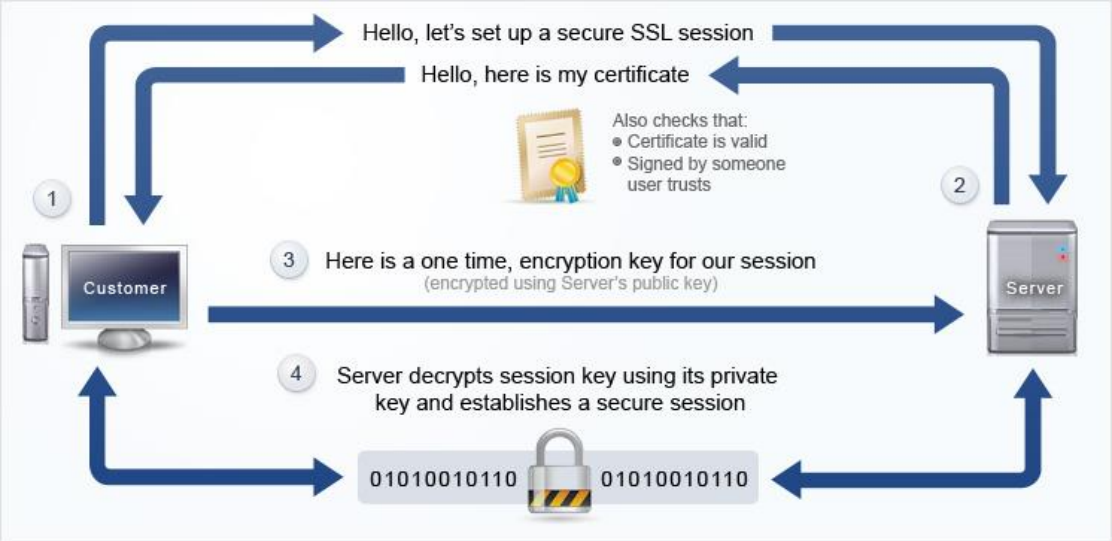
HTTPS on internetissä yleisesti käytetty salausmenetelmä, joka pohjautuu HTTP-protokollaan. HTTP on verkkoliikenteessä käytetty protokolla, jonka pohjalta muun muassa verkkosivut toimivat. HTTPS ei ole sinänsä itsenäinen protokolla, vaan se on HTTP-protokollan turvallinen variaatio. HTTPS tarvitsee toisen protokollan, joka salaa itse liikenteen. Tämänlaisia ovat TLS sekä sen edeltäjä SSL. Yhdessä HTTPS sekä TLS/SSL voivat tehdä liikenteestä salattua. (What is hypertext transfer protocol secure? n.d.)

2.1.2 SSL

SSL on protokolla, jonka avulla voidaan salata tietoliikenne selaimen ja palvelimen välillä. SSL-protokolla hyödyntää salauksen luodessaan julkisen avaimen ja symmetrisen avaimen yhdistelmää eli niin kutsuttua public-private key -paria. Tämä perustuu siihen, että julkisella avaimella salattu paketti voidaan purkaa vain sen yksityisellä avainparilla. SSL-protokollan avulla palvelin myös tunnistautuu käyttäjän selaimelle.

Tunnistautumisella varmistetaan se, että palvelin on todellakin se kuka se väittää olevansa. Näin ollen selain voi luottaa palvelimeen ja luoda salatun liikenteen palvelimen kanssa. (What is SSL 2016.)

Kuviossa 2 on kuvattuna SSL-salauksen vaiheet ja yhteyden muodostuminen asiakaslaitteen ja palvelimen välillä. Ensimmäiseksi asiakaslaite lähettää ”clienthello”-viestin palvelimelle, jossa asiakaslaite pyytää turvallista yhteyttä. Tämän jälkeen palvelin vastaa asiakaslaitteelle ”serverhello”-viestillä. Tähän viestiin sisältyvät palvelimen käyttämä sertifikaatti sekä julkinen avain, jotka asiakaslaite vastaanottaa sertifikaatin ja julkisen avaimen. Tämän jälkeen asiakaslaite lähettää salausavaimen sessiota varten ja salaa sen käyttäen palvelimen julkista avainta. Palvelin sitten purkaa asiakaslaitteen lähettämän sessioavaimen salauksen käyttämällä omaa yksityistä avaintaan. Purkamisen jälkeen palvelin muodostaa sessioavaimen avulla salatun yhteyden asiakaslaitteen ja itsensä välille. (Lim 2016.)



Kuvio 2. SSL-salauksen muodostuminen (SSL how it works 2009)

2.1.3 TLS

TLS pohjautuu SSL-protokollaan ja onkin siitä paranneltu versio. TLS:ää aloitettiin kehittämään, jotta saataisiin SSL:stä standardoitu versio. Näin SSL-versiosta 3.1 muodostui TLS 1.0. Tällä hetkellä on yleisesti käytössä TLS 1.2 -versio mutta TLS 1.3 -versio on jo kehitetty ja sekin on jossain muodossa käytössä. (Olenski 2016.)

TLS-protokollan hyöty verrattuna SSL-protokollaan nähden on sen turvallisuus. TLS-protokollaa on kehitetty eteenpäin SSL-protokollasta tekemällä korjauksia turvallisuuteen liittyen. (Kangas 2016.)

2.1.4 Julkisen avaimen kiinnittäminen

Julkisen avaimen kiinnittäminen (eng. public key pinning) on tekniikka, jonka avulla voidaan suojautua paremmin mahdollisilta MITM-hyökkäyksiltä. Julkisen avaimen kiinnittämisellä voidaan varmistaa yhdistäminen oikealle palvelimelle. Julkisen avaimen kiinnittäminen perustuu siihen, että palvelimen sertifikaatista saadaan CA:n julkinen avain, joka kerrotaan selaimelle. Selain tämän jälkeen yhdistäessä palvelimelle vertaa palvelimelta tulevaa julkista avainta, joka sijaitsee sertifikaatissa, tiedossa olevaan julkiseen avaimeen. Mikäli nämä täsmäävät, voidaan yhteys sallia. Muutoin yhteys hylätään, sillä ainoastaan tunnetuista lähteistä tulevat yhteydet ovat sallittuja. (Johansen 2015.)

2.1.5 Sertifikaattin kiinnittäminen

Sertifikaatin kiinnittäminen on tekniikaltaan samankaltainen kuin julkisen avaimen kiinnittäminen. Nimensä mukaisesti tällä kertaa selaimelle kerrotaan, minkälainen sertifikaatti pitäisi olla odotettavissa. Sertifikaatin kiinnittämisessä erona on se, että mikäli palvelimen sertifikaatti vaihtuu, se täytyy uudestaan kertoa selaimelle. Julkisen avaimen kiinnittämisessä ei tule tämänlaista ongelmaa, sillä yleensä sertifikaattien julkiset avaimet eivät vaihdu, mikäli uusiminen tehdään oikein. (Certificate pinning 2016.)

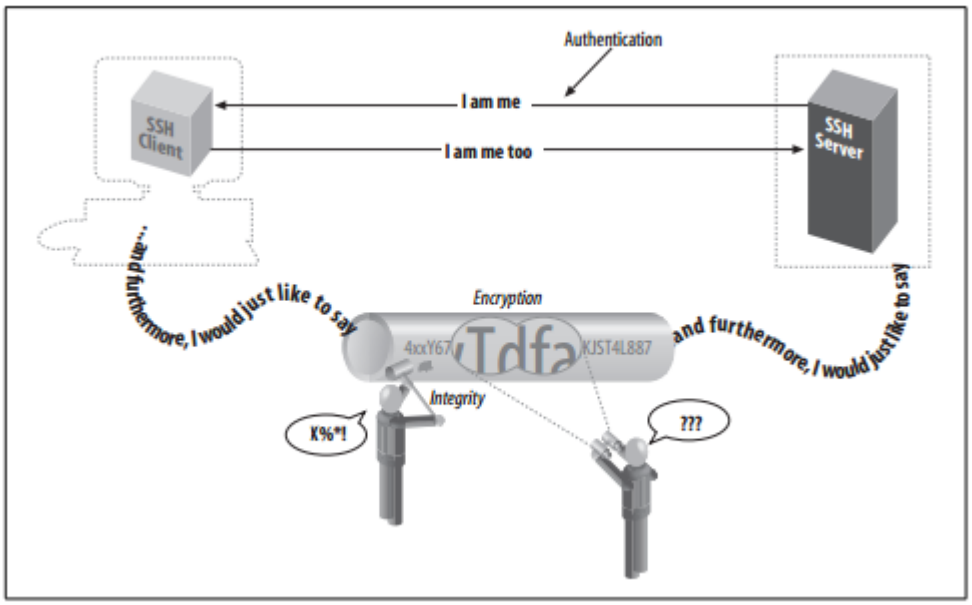
2.2 Tunnelointiprotokollat

HTTPS-salauksen toimiessa sertifikaattipohjaisesti on olemassa myös niin kutsuttuja tunnelointiprotokollia. Yleisesti tunneliprotokollista käytetään nimitystä VPN. VPN käyttää hyväkseen virtuaalista tunnelia, jonka sisällä dataa voidaan siirtää julkisen verkon ylitse salattuna. VPN-tunnelointi perustuu kapselointiin, jossa paketit tai kehykset sijoitetaan toisten pakettien tai kehysten sisään. Tällä tavalla kapseloituna paketin siirtäminen laitteelta toiseen on turvallista esimerkiksi internetin ylitse. (Nikkonen 2010.)

2.2.1 SSH

SSH on protokolla, jonka avulla voidaan luoda salaus käyttäjän koneen sekä etätietokoneen välille. SSH on yleisesti käytössä maailmalla. Sen avulla voidaan muodostaa tunnelointi. SSH on ensisijaisesti tarkoitettu Unix ja Linux -käyttöjärjestelmien etähallintaan. (Rouse 2016.)

Kuviossa 3 on kuvattuna SSH:n toimintaperiaate. Kuten kuviosta käy ilmi, SSH:n avulla voidaan tehdä salausta sekä tunnistautumista. Tunnistautuminen tapahtuu sekä käyttäjän että palvelimen osalta, joka parantaa tietoturvaa.



Kuvio 3. SSHn yleiskuvaus (Barret, Byrnes & Silverman 2001)

2.2.2 IPSec

IPSec itsessään ei ole protokolla vaan kokonaisuus määriteltyjä protokollia IETF:n toimesta, joiden avulla voidaan muodostaa IPSec-tietoturva. IPSec toimii kahdessa eri moodissa, joita ovat kuljetus- ja tunnelimoodi. IPSec suojaa IP-paketin riippuen moodista. Tunnelimoodissa IP-paketista suojataan koko IP-datagrammi. Kuljetusmoodissa taas suojataan ainoastaan hyötydata sekä osa otsikkokentistä. (Alakoski 2003.)

2.2.3 SSL VPN

SSL VPN on protokolla, jonka avulla voidaan muodostaa tunneli käyttäjän ja palvelimen välille. Tunnelin avulla voidaan suojata data ja tunnistaa käyttäjä ja palvelin luotettavasti. SSL VPN nimensä veroisesti käyttää SSL-protokollasta tuttua salausta. SSL VPN toimii OSI-mallin 4. kerroksesta ylöspäin, jolloin se eroaa muista käytössä olevista VPN toteutuksista. SSL VPN voidaan toteuttaa SSL-protokollan avulla sertifikaattipohjaisesti. (Shinder 2005.)

3 Autentikointi

Autentikointi on tietotekniikassa käytetty käsite. Käsitteellä tarkoitetaan varmistusta siitä, että joku on juurikin se, kuka se väittää olevansa. Autentikointi voidaan jakaa kolmeen päätekijään, joiden avulla voidaan autentikointi suorittaa luotettavasti. Ne ovat *mitä tiedät*, *mitä omistat* ja *mitä olet*. (Miessler 2005.)

Mitä tiedät on autentikoinnissa käytetty käsite. Se tarkoittaa, että henkilö itse tietää jotain sellaista mitä kukaan muu ei tiedä. Tämänlainen voi olla esimerkiksi salasana, käyttäjätunnus, pin-koodi tai jokin muu salainen tieto. Tämän tiedon avulla käyttäjä voidaan tunnistaa luotettavasti. Tämä perustuu siihen, että kukaan muu ei tiedä mitä käyttäjä tietää. (Rouse 2015.)

Mitä omistat käsitteellä tarkoitetaan mitä henkilö omistaa tai hallitsee. Tämä voi esimerkiksi olla jokin laite, mobiililaite tai vastaava. Tämäkin perustuu siihen, että ainoastaan yhdellä henkilöllä voi olla tämä laite hallussaan. Laite voidaan yksilöidä niin, ettei samanlaista laitetta voi olla kenelläkään muulla hallussa. Esimerkiksi matkapuhelin voidaan IMEI-koodin perusteella tunnistaa yksilöllisesti. (What are knowledge factors, possession factors and inherence factors? 2016.)

Mitä olet on autentikoinnissa käsite, joka tarkoittaa henkilön ominaisuuksia. Jokaisella henkilöllä on omat biometriset tunnisteet, joiden avulla henkilö voidaan yksilöidä luotettavasti. Näitä ovat DNA, silmät, sormenjäljet, kasvonpiirteet ja puheäänet. (What is multi-factor authentication? n.d.)

4 Tiedon eheys

4.1 Pariteettivarmistus

Pariteettivarmistus on tiedonsiirron tarkastuskeino, jonka avulla voidaan varmistaa siirretyn tiedon muuttumattomuus. Pariteettivarmistus toimii tiedonsiirrossa niin, että lisätään lähetettävän datan perään ylimääräinen bitti, joka ilmaisee, onko datassa parillinen vai pariton määrä bittejä. Mikäli data on muuttunut, ei tämä tarkastusbitti vastaa bittien määrää datassa. Esimerkiksi parillinen pariteettibitti ilmaisee, että datassa on parillinen määrä ykkösiä. Mikäli data saapuu niin, että ykkösiä on pariton määrä, tiedetään, että data on muuttunut. Huonona puolena pariteettivarmistuksessa on se, että voidaan varmistaa tiedon eheys vain, mikäli yksi bitti on muuttunut tai pariton määrä bittejä muuttuu. (Hassinen, Malinen & Miettinen 2001.)

4.2 Tarkistussumma

Tarkistussumma on tiedon eheyden tarkastukseen käytetty summa, joka muodostuu tarkastussumma algoritmin avulla. Yksinkertaistettuna datasta muodostetaan eräänlaisella algoritmilla tiiviste ja tämä tiiviste voidaan lähettää datan mukana. Vastaanottaessa data voidaan laskea datasta uudestaan samalla algoritmilla tiiviste ja jos se täsmää alkuperäisen tiivisteen kanssa, tiedetään, ettei data ole muuttunut lähetyksessä. (Fisher 2016.)

5 Mitä salauksella saavutetaan?

Verkkoliikenteen salaus on tärkeää, sillä verkkorikollisuus on lisääntynyt vuosien kuluessa. Etenkin mobiililaitteisiin kohdistunut uhka on kasvanut merkittävästi. Tiedon urkinta on myös kohdistumassa yhä useammin tavalliseen käyttäjään. (The relentless growth of cybercrime 2016.)

5.1 Hyödyt

Salauksella voidaan estää ulkopuolisten mahdollisuus lukea verkkoliikennettä. Varsinkin HTTPS-salauksen avulla voidaan saada hyötyä muun muassa palvelimen tunnistuksesta. Sen avulla varmistetaan, että palvelin on juurikin se, mikä sen halutaan olevan. Mikään salaus ei ole hyödyllinen, mikäli toinen osapuoli on hyökkääjän hallussa tai yhdistetään väärään palvelimeen. (Vaughan 2014.)

5.2 Haitat

Salaukseen liittyy aina huonoja puolia. Sertifikaattipohjaisissa ratkaisuissa täytyy pitää mielessä, että sertifikaatit ovat yleensä maksullisia. On olemassa kuitenkin saatavilla sertifikaatteja ilmaiseksi joidenkin myöntäjien toimesta. Lisäksi on mahdollista tehdä sertifikaatteja itse, mutta tällöin menetetään mahdollisesti palvelimen luotettavuus. Yksi ongelma on myös salauksen vaatima teho palvelimelta. Salauksen käyttö voi kuormittaa palvelimia ja viedä laskentatehoa, jonka vuoksi voi esiintyä ajoittaista hitautta palveluissa. (Finley 2014.)

6 Vertailu

Jokaisella salausmenetelmällä on hyviä ja huonoja ominaisuuksia. Salausmenetelmiä ei välttämättä voida täysin suoraan verrata keskenään, sillä jokaisella niistä on olemassa omat vaatimuksensa käytettävissä olevan laitteiston osalta. Vertailua voidaan kuitenkin tehdä ottamalla huomioon protokollien mahdolliset vaatimukset sekä ominaisuudet.

6.1 HTTPS

HTTPS on luotettava ja turvallinen salausmenetelmä. Sen avulla voidaan luoda vahva salaus sekä tunnistaa luotettavasti palvelin. Lisäksi HTTPS on yleisesti tunnettu protokolla, ja tämän vuoksi se on tuettu erittäin hyvin. Mobiiliapplikaation ollessa selain pohjainen on sen käyttöönotto ja implementointi järjestelmään melko helppoa.

HTTPS voi mahdollisesti osoittautua kalliimmaksi kuin mahdolliset muut salaukset, sillä HTTPS vaatii toimiakseen sertifikaatin asennettuna palvelimelle. On mahdollista

tehdä sertifikaatit itse, mutta se ei ole haluttua mahdollisten sertifikaattiepäluotta-musilmoitusten vuoksi. Nämä ilmoitukset ovat kuitenkin mahdollista kiertää teknisin ratkaisuin, esimerkiksi asentamalla tarvittavat juurisertifikaatit laitteille.

6.2 Tunnelointiprotokollat

Tunneloinnin avulla voidaan muodostaa erittäin hyvä salaus sekä tunnistaa sekä asiakaslaite että palvelin. Lisäksi nämä protokollat eivät vaadi mitään kallista ratkaisua vaan yleensä ainoastaan konfiguraatiomuutoksia järjestelmiin.

Huonona puolena tunnelointiprotokollat vaativat yleensä resursseja niin palvelimelta kuin asiakaslaitteelta. Lisäksi tunnelointiprotokollat ovat teknisempiä kuin esimerkiksi HTTPS-yhteys, jossa käyttäjän ei tarvitse tehdä mitään.

7 Hyökkäystavat lähiverkossa

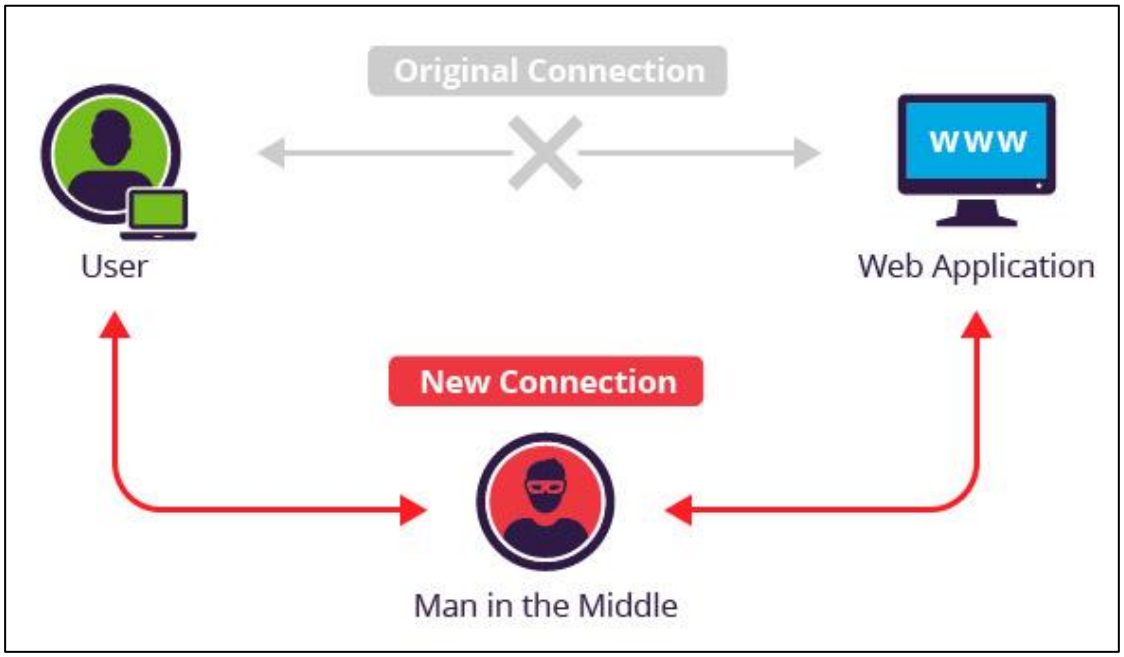
Ilman suojausta voidaan toteuttaa useita hyökkäyksiä salaamatonta liikennettä kohtaan. Hyökkäyksien tekeminen on nykypäivänä kohtuullisen helppoa ja nopeaa. Automaattisilla työkaluilla hyökkäyksien toteuttaminen ei vaadi juurikaan teknistä osaamista tai tietoa.

7.1 Man in the middle

7.1.1 Yleistä

Man in the middle tai toisin sanoen MiTM on hyökkäystapa, jolla saadaan kaapattua verkossa kulkevaa liikennettä omaan tarkasteluun. Tarkastelun jälkeen liikenne voidaan halutessa lähettää eteenpäin joko muutettuna tai alkuperäisessä muodossa. Liikenteestä voidaan mahdollisesti haluta muun muassa käyttäjätunnuksia sekä salasanoja. (Fisher 2013.)

Kuviossa 4 on kuvattuna pääperiaatteittain MiTM hyökkäys. Siinä hyökkääjä on muodostanut uuden yhteyden käyttäjän ja palvelimen välille toimien itse ”välityspalvelimena”. MitM hyökkäys voi koostua monesta eri hyökkäyksestä.



Kuvio 4. Man in the middle -hyökkäys (Man in the Middle (MITM) Attack. n.d.)

7.1.2 ARP poisoning

ARP poisoning eli ARP-taulun myrkytys on hyökkäystapa, jonka avulla voidaan kaapata liikennettä lähiverkossa toimivien laitteiden välillä. Hyökkäys tapahtuu myrkyttämällä hyökättävän laitteen ARP-taulu niin, että sinne muodostuu väärennettyjä IP- ja MAC-osoitepareja. Näiden väärennettyjen parien avulla hyökkäyksen kohteena oleva laite lähettää viestinsä oikean IP-osoitteen perusteella, mutta MAC väärennöksen vuoksi se päätyykin hyökkääjälle. (ARP poisoning attack and mitigation techniques 2016.)

Kuviossa 5 nähdään ARP-taulu niin kuin se kuuluisi olla. Taulussa nähdään kaksi eri IP-osoitetta ja niitä vastaavat MAC-osoitteet.

```
Interface: 172.16.13.76 --- 0x5
  Internet Address      Physical Address      Type
  172.16.13.73          ec-8e-b5-46-0d-88     dynamic
  172.16.13.77          00-07-50-f5-e1-00     dynamic
```

Kuvio 5. ARP-taulu ennen myrkytystä

Kuviossa 6 nähdään sama ARP-taulu myrkytyksen jälkeen. Taulussa näkyvät edelleen samat IP-osoitteet, mutta MAC-osoite on kummallakin IP-osoitteella sama.

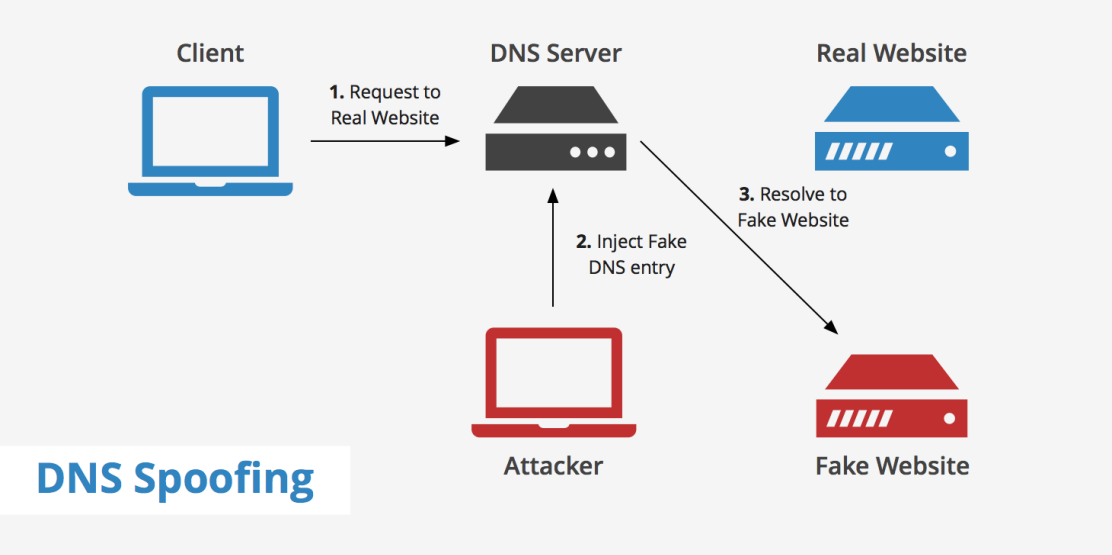
```
Interface: 172.16.13.76 --- 0x5
  Internet Address      Physical Address      Type
  172.16.13.73          ec-8e-b5-52-55-82     dynamic
  172.16.13.77          ec-8e-b5-52-55-82     dynamic
```

Kuvio 6. ARP-taulu myrkytyksen jälkeen

Kahden IP-osoitteen takaa on näkyvissä sama MAC-osoite. Tämän tilanteen vuoksi oikeaan IP-osoitteeseen lähetettäessä päätyykin viesti väärennettyyn MAC-osoitteeseen. Tällöin hyökkääjä voi tarkastella lähetettyjä viestejä.

7.1.3 DNS spoofing

DNS spooffaus on käsite, joka liittyy MITM-hyökkäyksiin. Siinä tarkoituksena on väärentää uhrin DNS-palvelut niin, että uhri ohjataan väärään internetosoitteeseen. DNS-kyselyt tapahtuvat väärennetylle DNS-palvelimelle, jolloin hyökkääjä voi itse päättää, kuinka uhria ohjataan eri sivuille. Kuviossa 7 on toteutettu DNS spooffaus, jossa DNS-palvelimeen on injektoitu vääriä DNS-merkintöjä, jolloin uhri ohjautuu väärälle verkkosivulle. (What is DNS spoofing? 2017.)



Kuvio 7. DNS spooffaus (What is DNS spoofing? 2017)

8 Toteutus

8.1 Mihin päädyttiin

Työn tutkimusten pohjalta päädyttiin ryhtyä tutkimaan ja toteuttamaan mahdollista HTTPS-salausta liikenteelle. HTTPS-valintaan vaikutti se, että se on protokollana kevyt niin mobiililaitteelle kuin palvelimelle. Se ei myöskään vaadi erityisiä konfiguraatioita käyttäjältä vaan toimii huomaamattomasti taustalla. Lisäksi se on tuettuna kaikilla mobiilialustoilla ja soveltuu parhaiten hybrid-mobiiliapplikaatiolle, joka toimii selainpohjaisesti.

8.2 Ympäristöt

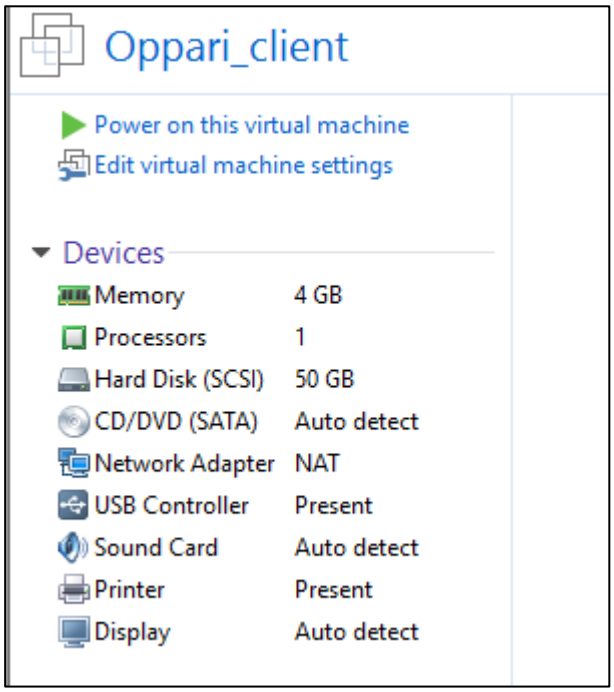
HTTPS-salauksen testausta varten pystytettiin ympäristö, jossa testiä voitiin suorittaa ja löytää mahdollisia ongelmia käytännön tasolla. Testauksessa simuloitiin toimivaa ympäristöä luomalla virtualisoituja koneita VMware Workstation 12 -ohjelmaan. Näiden lisäksi käytössä oli Android-käyttöjärjestelmällä oleva mobiililaite, jolla sovelluksen kautta yhdistämistä palvelimeen voitiin testata.

Mobiiliympäristö luotiin virtualisoidulle Linux-tietokoneelle, jossa käyttöjärjestelmänä oli Ubuntu. Linux-tietokoneelle asennettiin Android-alustan testaukseen vaadittavat ohjelmistot. Itse mobiilisovellusta rakennettiin Cordova-alustalla, jolla voidaan emuloida tarvittaessa itse mobiililaitteen käyttäytymistä. Lisäksi käytössä oli Android-käyttöjärjestelmällä oleva mobiililaite, jolla testattiin yhteyttä sovelluksen välityksellä palvelimeen.

Liikennöintiä varten luotiin apache2-pohjainen verkkopalvelin. Verkkopalvelin toimi Linux-palvelimella, jossa käyttöjärjestelmänä oli Ubuntu. Apachen avulla luotiin yksinkertainen verkkosivu, jossa toteuttiin SSL-salausta HTTPS-yhteydellä. Palvelimelle luotiin OpenSSL:n avulla sertifikaatti, jota käytettiin salatun yhteyden luomiseen. Sertifikaatti tehtiin niin kutsuttuna ”multi-domain sertifikaattina”, jossa voidaan määritellä useampia sallittuja nimiä yhteyden muodostamiseen.

8.3 Mobiiliympäristön asennus

Mobiiliympäristö asennettiin VMware Workstation 12 -ohjelmaan oletusasetuksilla. Muistia järjestelmälle annettiin 4 GB, jotta mahdolliset simulaatiot toimisivat nopeammin. Kuviossa 8 on kuvattuna koneen asetukset VMware Workstationissa.



Kuvio 8. Mobiiliympäristön koneen asetukset

Aluksi ennen itse Cordovan asennusta täytyy varmistaa tarvittavien ohjelmien asennus. Nodejs tarvitaan Cordovaa varten ja se asennetaan seuraavalla komennolla:

sudo apt-get install nodejs

Lisäksi täytyy nodejs:lle lisätä alias node, jonka avulla cordova osaa käyttää sitä:

sudo ln -s /usr/bin/nodejs /usr/bin/node

Onnistunut nodejs asennus voidaan todentaa kuvion 9 mukaisella komennolla, joka tulostaa asennetun nodejs version.

```
toor@toor:~$ nodejs --version
v4.2.6
```

Kuvio 9. nodejs asennuksen varmistaminen

Cordova tarvitsee asentuakseen node package managerin (NPM), joka asennetaan komennolla:

sudo apt-get install npm

Tämän npm:n asennus voidaan varmistaa kuvion 10 mukaisesti.

```
toor@toor:~$ npm --version
3.5.2
```

Kuvio 10. npm asennuksen varmistus

Seuraavaksi tarvitsee asentaa git, jota Cordova käyttää taustalla toimiakseen. Sen asennus tapahtuu komennolla:

sudo apt-get install git

Git asennuksen voi varmistaa jälleen kuvion 11 komennolla.

```
toor@toor:~$ git --version
git version 2.7.4
```

Kuvio 11. Git asennuksen varmistaminen

Tämän jälkeen itse Cordova voidaan asentaa npm:ää käyttäen komennolla:

sudo npm install -g cordova

Tämän jälkeen varmistetaan onnistuneen Cordovan asennus kuvion 12 komennolla. Mikäli tulosteena on Cordovan version numero, se tarkoittaa, että asennus on tällöin onnistunut.

```
toor@toor:~$ cordova --version
7.0.1
```

Kuvio 12. Cordovan versio ja asennuksen varmistus

Cordovan asennuksen jälkeen tarvitaan Android studio ja sen käyttämät työkalut. Android studion avulla voidaan testata sovelluksia Android käyttöjärjestelmällä.

Työssä käytetään ainoastaan Android studion SDK-työkaluja. Android studio tarvitsee toimiakseen Javan, ja sekä Java että Android studio tarvitsevat 64-bittisessä Linux versiossa omat lisäyksensä kirjastoihin. Lisäyksien asentaminen tapahtuu seuraavalla komennolla:

sudo apt-get install -qq libc6:i386 libgcc1:i386 libstdc++6:i386 libz1:i386

Javan asentaminen tapahtuu lataamalla Java SE development kit. Lataaminen tapahtuu http://www.oracle.com/technetwork/java/javase/downloads/index.html -osoitteesta. Lataamisen jälkeen ajetaan asentamiseen tarvittavat komennot:

mkdir -p ~/java

cd ~/Downloads

mv jdk-8u131-linux-x64.tar.gz ~/java

cd ~/java

tar xvzf jdk-8u131-linux-x64.tar.gz

Onnistuneen Java SE development kitin asentamisen jälkeen tarvitsee asentaa Apache Ant, joka on Javan käyttämä ohjelmisto ohjelmistojen ajamiseen. Apache Ant voidaan ladata http://ant.apache.org/bindownload.cgi osoitteesta, jonka jälkeen Apache Ant:in asentaminen tapahtuu seuraavin komennoin:

mkdir -p ~/ant

cd ~/ant

mv ~/Downloads/apache-ant-1.10.1-bin.zip ~/ant

unzip apache-ant-1.10.1-bin.zip

Seuraavaksi asennetaan Android SDK työkalut, jotka ladataan http://developer.android.com/sdk -osoitteesta. Sieltä valitaan käyttöjärjestelmän mukainen sdk-paketti. Työkalujen asennus tapahtuu seuraavasti:

cd

mv ~/Downloads/sdk-tools-linux-3859397.zip .

unzip sdk-tools-linux-3859397.zip

Vielä tarvitsee asentaa ympäristö muuttujat ~/.bashrc-tiedostoon, jotta asennetut ohjelmat toimivat oikein. Kuviossa 13 on muuttujat tallennettu tiedostoon.

```
JAVA_HOME=~/java/jdk1.8.0_131
PATH=$PATH:~/java/jdk1.8.0_131/bin
export JAVA_HOME

ANT_HOME=~/ant/apache-ant-1.10.1
PATH=$PATH:~/ant/apache-ant-1.10.1/bin
export ANT_HOME

PATH=$PATH:~/platform-tools
PATH=$PATH:~/tools

export PATH
```

Kuvio 13. Ympäristömuuttujat

Komennot vaativat automaattisesti toimiakseen .bash_profile -tiedoston, minne lisätään kuvion 14 mukaiset rivit.

```
if [ -f ~/.bashrc ]; then
        source ~/.bashrc
fi
```

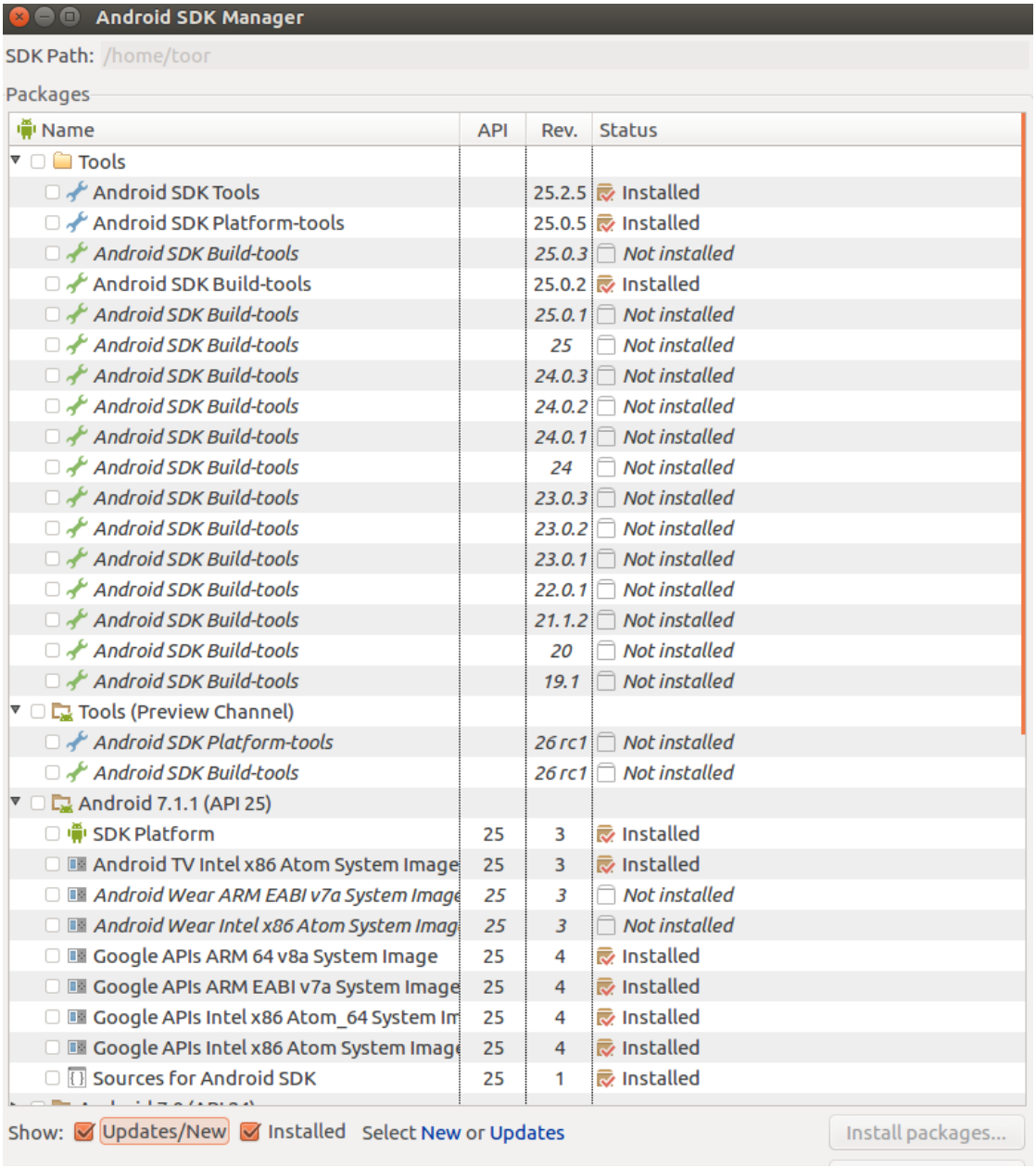
Kuvio 14. .bash_profile-tiedoston sisältö

Tämän jälkeen voidaan varmistaa asennettujen sovellusten toiminta kuvion 15 mukaisin komennoin. Josta viimeinen komento ”android” käynnistää Android SDK manager -ohjelman.

```
toor@toor:~$ javac -version
javac 1.8.0_121
toor@toor:~$ java -version
java version "1.8.0_121"
Java(TM) SE Runtime Environment (build 1.8.0_121-b13)
Java HotSpot(TM) 64-Bit Server VM (build 25.121-b13, mixed mode)
toor@toor:~$ ant -version
Apache Ant(TM) version 1.10.1 compiled on February 2 2017
toor@toor:~$ android
```

Kuvio 15. Asennuksien testaaminen

Android komennolla saadaan käynnistettyä Android SDK manager, josta voidaan asentaa tarvittavat työkalut. Työkalujen avulla voidaan simuloida Android käyttöjärjestelmää erittäin hyvin. Kuviossa 16 on kuvattu Android SDK manager.



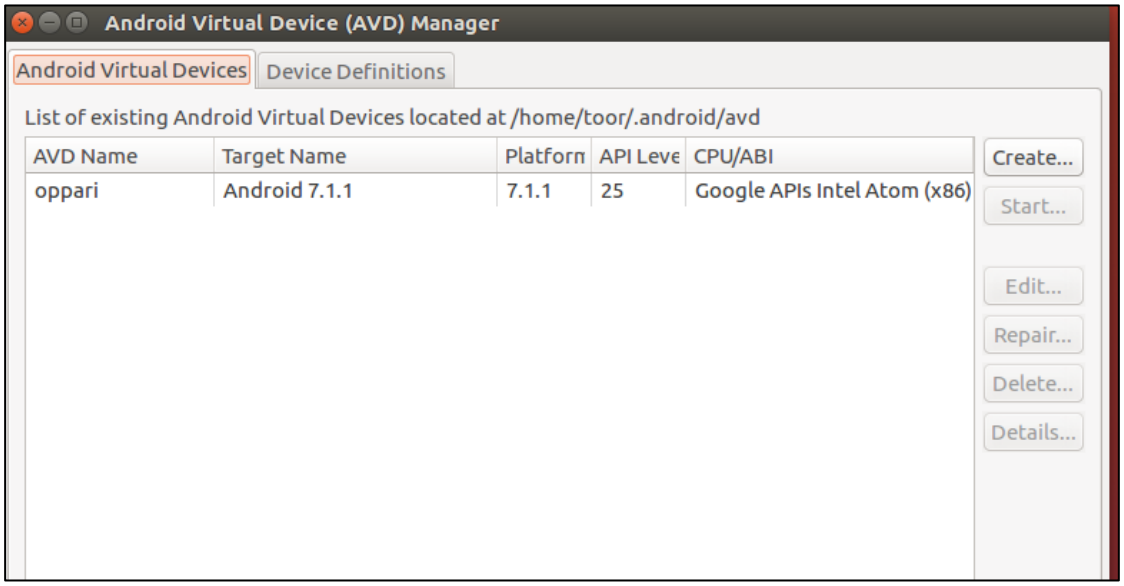
Kuvio 16. Android SDK manager -ohjelma

Tarvittavia työkaluja Android-käyttöjärjestelmän simulointiin ovat jo asennetut Android SDK Tools, Android SDK Platform-tools, Android SDK Build-tools sekä itse Android-käyttöjärjestelmä Android 7.1.1.

Käyttöjärjestelmän simulointia varten täytyy luoda tarvittava ympäristö. Ympäristö pääsee luomaan Android Virtual Device manager -ohjelmalla ja se käynnistyy komennolla:

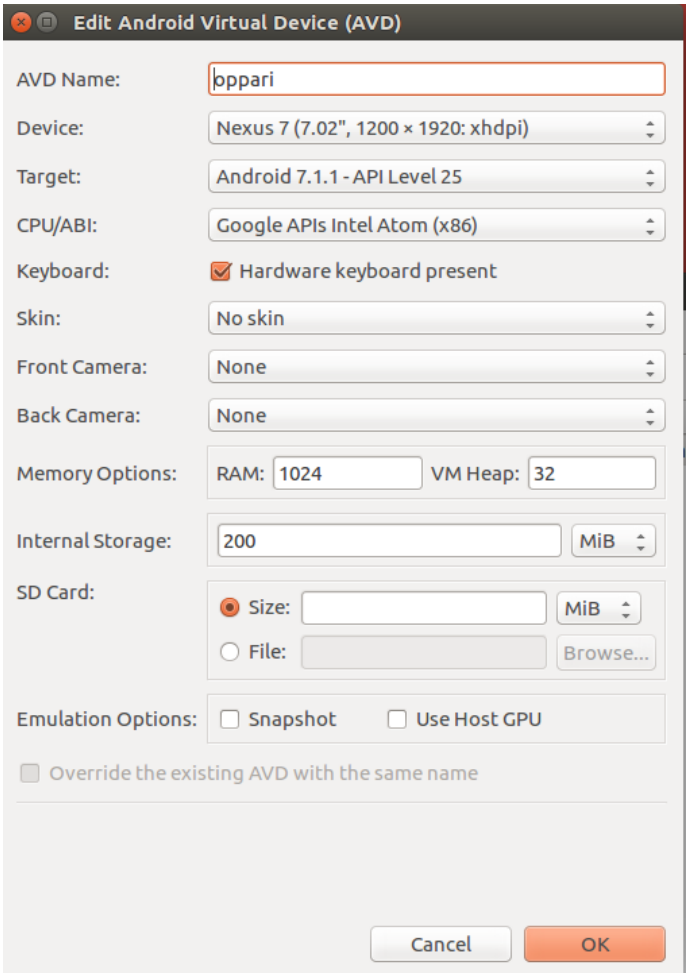
android avd

Tätä opinnäytetyötä varten on luotu android 7.1.1 -käyttöjärjestelmällä toimiva emulaattori. Kuviossa 17 nähdään luotu käyttöjärjestelmä.



Kuvio 17. Android Virtual Device Manager

Asetuksista päästään muokkaamaan muun muassa RAM-muistin määrää, minkälainen CPU-yksikkö on kyseessä ja puhelimen mallia. Kuviossa 18 on kuvattuna asetukset tässä työssä käytettyyn emulaattoriin.

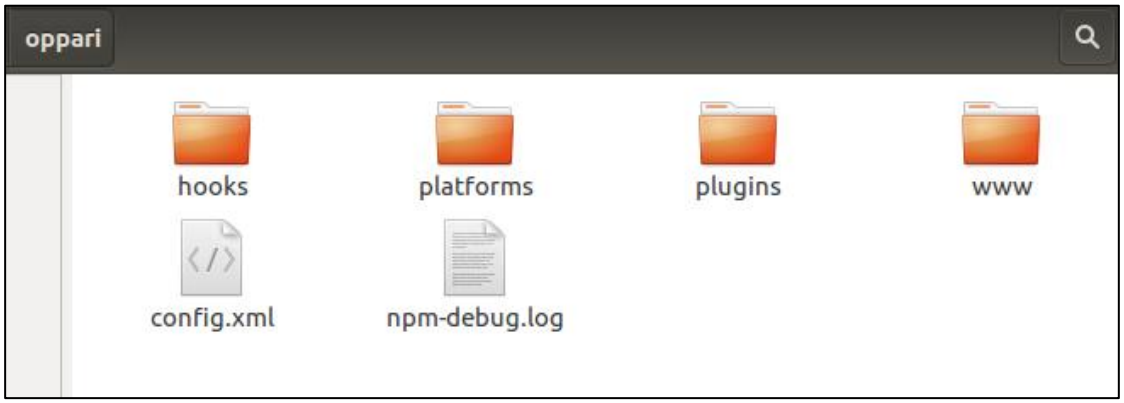


Kuvio 18. Emulaattoriasetukset

Asetuksien ollessa kunnossa voidaan mobiiliapplikaation rakentaminen aloittaa. Työssä tehtiin yksinkertainen opinnäytetyö-sovellus, jota voidaan muokata omien halujen mukaan. Projektin luominen Cordovan avulla on helppoa ja tehtiin komennolla:

cordova create oppari com.oppari Oppari

Komento luo yksinkertaisen perusrakenteen mobiiliapplikaatiota varten. Kuviossa 19 on kuvattuna tarkemmin mitä komennolla luotiin.



Kuvio 19. Cordova projektin luonti

Tärkeimmät ovat config.xml, plugins, platforms sekä www. Config.xml on globaali konfiguraatiotiedosto, jonka avulla voidaan muun muassa ”whitelistata” verkko-osoitteita. Plugins-kansiossa sijaitsevat applikaatiossa käytettävät pluginit. Platforms-kansio sisältää ajettavat tiedostot joita eri alustat kuten esimerkiksi Android sekä IOS tarvitsevat. Viimeinen eli www-kansio sisältää applikaation tiedostot, joiden avulla rakennetaan sovellus. Sovellusta muokatessa tehdään komentoja www-kansion sisällä oleviin tiedostoihin kuten javascript-tiedostoon. Kuviossa 20 on kuvattuna javascript-tiedosto, joka sijaitsee www-kansion alla js-kansiossa.

index.js (~/oppari/www/js) - gedit

Open

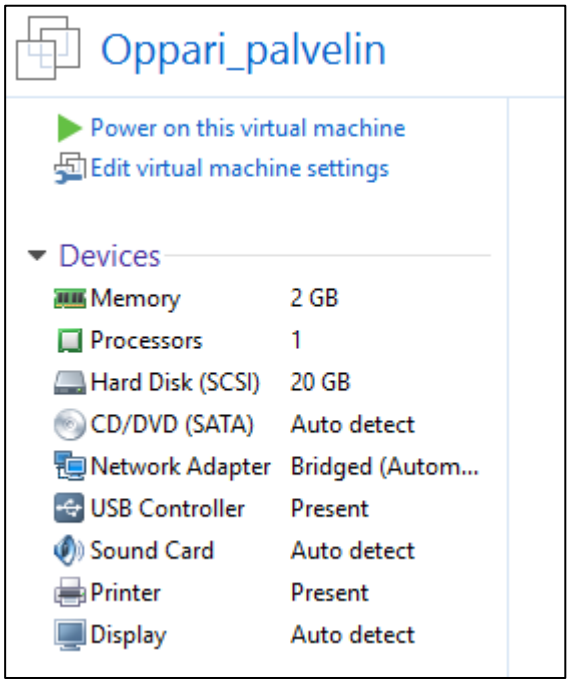
```
var app = {
    // Application Constructor
    initialize: function() {
        document.addEventListener('deviceready', this.onDeviceReady.bind(this), false);
    },

    // deviceready Event Handler
    //
    // Bind any cordova events here. Common events are:
    // 'pause', 'resume', etc.
    onDeviceReady: function() {
        this.receivedEvent('deviceready');
var ref = cordova.InAppBrowser.open('https://192.168.0.115', '_blank', 'location=yes');
    },

    // Update DOM on a Received Event
    receivedEvent: function(id) {
        var parentElement = document.getElementById(id);
        var listeningElement = parentElement.querySelector('.listening');
        var receivedElement = parentElement.querySelector('.received');

        listeningElement.setAttribute('style', 'display:none;');
        receivedElement.setAttribute('style', 'display:block;');

        console.log('Received Event: ' + id);
    }
};

app.initialize();
```

Kuvio 20. Js-tiedoston sisältö

Kuviosta 20 nähdään korostettuna sinne muutetut rivit. Korostetulla koodilla kutsutaan pluginia InAppBrowser:ia avautumaan osoitteeseen https://192.168.0.115. Avaus tapahtuu vasta kun tapahtuma ”deviceready” on valmis. Tällä varmistetaan, että applikaatio on valmis ennen kuin pluginia kutsutaan.

8.4 Palvelinympäristön pystytys

Palvelin asennettiin VMware Workstation 12 -alustalle oletusasetuksilla. Käyttöjärjestelmänä toimi Ubuntu 16.04 LTS, jossa oli graafinen käyttöliittymä. Kuviosta 21 käy ilmi palvelimen asetukset VMware Workstation 12 -ohjelmassa. Palvelimelle konfiguroitiin sillattu verkkoadapteri, jolloin palvelimelle voitiin antaa kiinteä IP-osoite samasta aliverkosta mistä reititin jakaa osoitteensa. Tällä tavalla saatiin matkapuhelin kommunikoimaan langattoman verkon kautta palvelimelle.



Kuvio 21. Palvelimen asetukset

Palvelimelle asennettiin ja konfiguroitiin apache2, jonka avulla voitiin tarjota omaa verkkosivustoa mobiiliapplikaatiolle. Asennus tapahtui komennolla:

sudo apt-get install apache2

Jonka jälkeen luotiin tiedostopolku /var/www/oppari.com/public_html/index.html sekä konfiguroitiin index.html-tiedosto näyttämään kuvion 22 mukaiselta.

```
<html>
  <head>
    <title>Welcome to oppari.com!</title>
  </head>
  <body>
    <h1>Success!  The oppari.com is working!</h1>
  </body>
</html>
```

Kuvio 22. Index.html tiedoston sisältö

Seuraavaksi konfiguroitiin itse sivusto saatavaksi. Tämä tapahtui konfiguroimalla tiedostopolkuun /etc/apache2/sites-available/ oppari.com.conf-tiedosto. Tiedostoon muokattiin kuvion 23 mukaiset konfiguraatiot.

```
<VirtualHost *:80>
    ServerAdmin admin@example.com
    ServerName oppari.com
    ServerAlias www.oppari.com
    DocumentRoot /var/www/oppari.com/public_html
    ErrorLog ${APACHE_LOG_DIR}/error.log
    CustomLog ${APACHE_LOG_DIR}/access.log combined
</VirtualHost>
```

Kuvio 23. Oppari.com.conf-tiedoston konfiguraatio

Tämän jälkeen seuraavilla komennoilla käynnistettiin ja aktivoitiin konfiguroidut www-sivut.

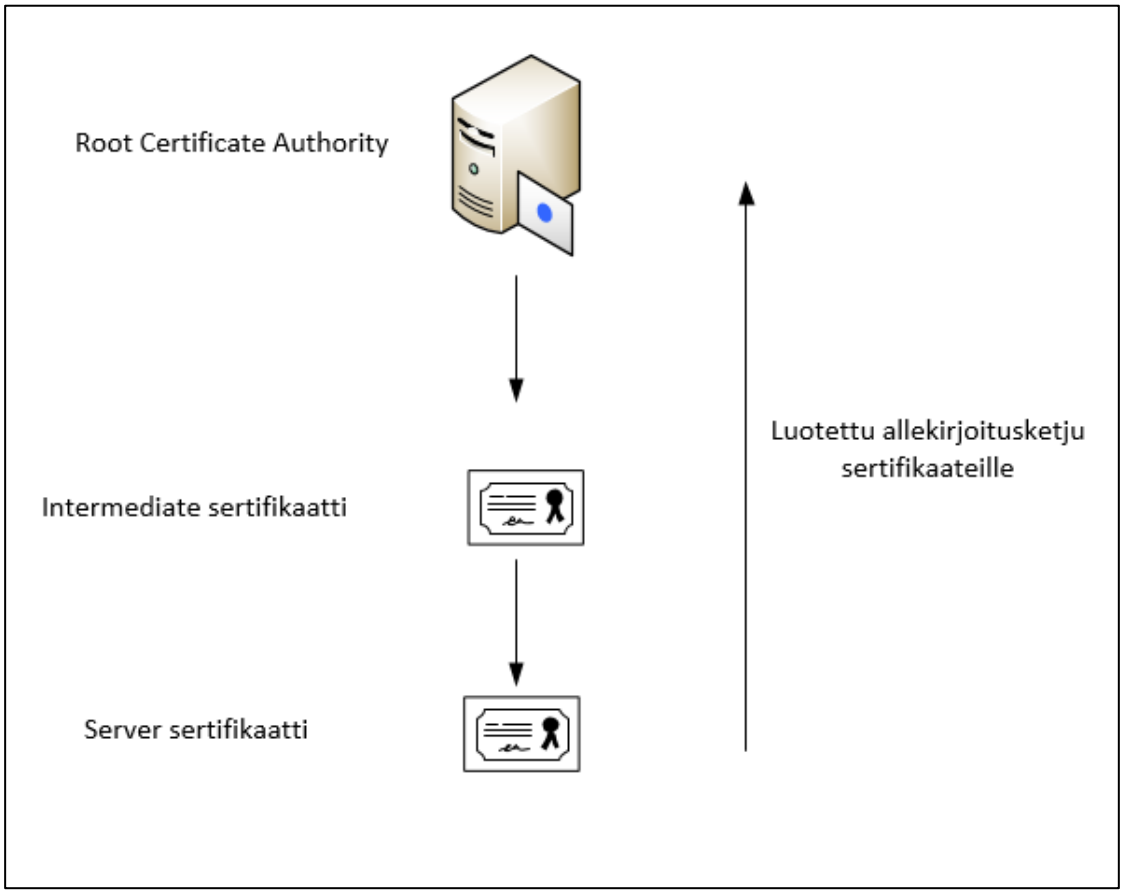
sudo a2ensite oppari.com.conf

sudo service apache2 restart

Tällä tavalla luotiin yksinkertainen web-palvelin, johon yhdistäessä näyttä tulosteen konfiguroidusta public_html-tiedostosta.

8.5 Sertifikaattit

HTTPS-salauksen ollessa sertifikaatteihin perustuva tarvitsee palvelin sertifikaatin. Sertifikaatit luotiin itse openssl-ohjelmalla. Sertifikaatteja varten luotiin palvelin, joka toimi juuri ”certificate authorityna”. Tämä juuri-palvelin allekirjoitti ja myönsi keskitason sertifikaatin, jonka avulla voitiin palvelimelle allekirjoittaa palvelimen tarvitsema sertifikaatti. Kuviossa 24 on kuvattuna sertifikaattiketju, jota käytettiin.



Kuvio 24. Sertifikaattiketju

Luotetun sertifikaattiketjun avulla voitiin mallintaa kuinka oikeat sertifikaatin myöntäjät toimivat. Lisäksi voitiin testata kuinka luotetulta myöntäjältä oleva sertifikaatti toimisi tämänlaisessa tilanteessa. Palvelimelle muista eroten luotiin niin kutsuttu multi-domain sertifikaatti. Tämänlaiselle sertifikaatille määritellään useampi vaihtoehtoinen nimi, joiden alla palvelin voi toimia niin, että sertifikaatti on luotettava. Palvelin voi tällöin toimia julkisella nimellä esimerkiksi www.oppari.com sekä samalla olla saavutettavissa yksityisenverkon IP-osoitteen 192.168.0.115 kautta.

8.5.1 Juurisertifikaatti

Sertifikaatit luotiin omalla palvelimella openssl-sovelluksella. Openssl-komentoja varten muokattiin Openssl:n konfiguraatio-tiedostoa, jonka pohjalta openssl-sovellus teki sertifikaatit. Openssl:lle määriteltiin konfiguraatio tiedosta (liite 1), jonka pohjalta Openssl muodosti sertifikaatin. Seuraavilla komennoilla luotiin juuri CA:n yksityinen avain ja sertifikaatti:

openssl genrsa -aes256 -out private/ca.key.pem 4096

openssl req -config openssl.cnf \

-key private/ca.key.pem \

-new -x509 -days 7300 -sha256 -extensions v3_ca \

-out certs/ca.cert.pem

Tämän jälkeen annettiin sertifikaatille tarvittavat tiedot kuvion 25 mukaisesti.

```
Country Name (2 letter code) []:FI
State or Province Name (full name) []:UUSIMAA
Locality Name (eg, city) []:ESPOO
Organization Name (eg, company) []:Oppari Oy
Organizational Unit Name (eg, section) []:
Common Name []:oppari.com
Email Address []:
```

Kuvio 25. Juurisertifikaatin tiedot

Näin luotiin juurisertifikaatti, joka toimii CA:na ja allekirjoittaa keskitason sertifikaatin.

8.5.2 Keskitason sertifikaatti

Juurisertifikaatin jälkeen tehtiin keskitason sertifikaatin yksityinen avain ja sertifikaatin allekirjoituspyyntö komennoilla:

openssl genrsa -aes256 -out intermediate/private/intermediate.key.pem 4096

openssl req -config intermediate/openssl.cnf -new -sha256 \

-key intermediate/private/intermediate.key.pem \

-out intermediate/csr/intermediate.csr.pem

Seuraavaksi täytetään tiedot samalla tavalla kuin aikaisemmin juurisertifikaatissa mutta muuttamalla “common name” -kentän nimeksi OPPARI INTERMEDIATE CA, kuten kuviossa 26 on nähtävissä.

```
Country Name (2 letter code) []:FI
State or Province Name (full name) []:UUSIMAA
Locality Name (eg, city) []:ESPOO
Organization Name (eg, company) []:Oppari Oy
Organizational Unit Name (eg, section) []:
Common Name []:INTERMEDIATE OPPARI CA
```

Kuvio 26. Keskitason sertifikaatin tiedot

Seuraavaksi tehdään keskitason sertifikaatille allekirjoituspyyntö juurisertifikaatilta. Openssl:lle määriteltiin konfiguraatiotiedosto (liite 2), jonka pohjalta se muodosti sertifikaatin. Allekirjoitus pyyntö tällä konfiguraatiotiedoston kanssa tapahtuu komennolla:

openssl ca -config openssl.cnf -extensions v3_intermediate_ca \

-days 3650 -notext -md sha256 \

-in intermediate/csr/intermediate.csr.pem

-out intermediate/certs/intedmediate.cert.pem

Tuloksena saatiin keskitason sertifikaatti, jonka oma CA on allekirjoittanut.

8.5.3 Palvelimen sertifikaatti

Seuraavaksi tehtiin allekirjoituspyyntö keskistason sertifikaatille palvelimen sertifikaattia varten. Sertifikaattia varten käytettiin konfiguraatiotiedostoa liite 2. Ensiksi täytyy luoda palvelimen yksityinen avain ja sen jälkeen allekirjoituspyyntö. Nämä tapahtuvat komennoilla:

openssl genrsa -aes256 -out intermediate/private/www.oppari.com.key.pem 2048

openssl req -config intermediate/openssl.cnf \

-key intermediate/private/www.oppari.com.key.pem \

-new -sha256 -out intermediate/csr/www.oppari.com.csr.pem

Täytetään kuvion 27 mukaisesti sertifikaatin tiedot.

```
Country Name (2 letter code) []:FI
State or Province Name (full name) []:UUSIMAA
Locality Name (eg, city) []:ESPOO
Organization Name (eg, company) []:OPPARI
Organizational Unit Name (eg, section) []:
Common Name []:www.oppari.com
Email Address []:
```

Kuvio 27. Palvelimen sertifikaatin tiedot

Common name -kenttään annetaan tässä vaiheessa palvelimen julkinen nimiosoite eli FQDN. Tämän tiedon täytyy täsmätä osoitteeseen, jolla yhdistetään palvelimeen. Koska palvelimelle halutaan usean domainin kanssa toimiva sertifikaatti, määritellään vaihtoehtoiset nimet openssl-ohjelman käyttämään konfiguraatiotiedostoon seuraavalla kuvion 28 osoittamalla tavalla.

```
[ server_cert ]
# Extensions for server certificates (`man x509v3_config`).
basicConstraints = CA:FALSE
nsCertType = server
nsComment = "OpenSSL Generated Server Certificate"
subjectKeyIdentifier = hash
authorityKeyIdentifier = keyid,issuer:always
keyUsage = critical, digitalSignature, keyEncipherment
extendedKeyUsage = serverAuth
subjectAltName = @alternate_names

[ alternate_names ]

DNS.1   = www.oppari.com
DNS.2   = testi.oppari.com
IP.1    = 192.168.0.115
```

Kuvio 28. Openssl konfiguraatiotiedosto

Määritetään serverin sertifikaatin myöntämiseen subjectAltName ja listataan halutut vaihtoehtoiset nimet alle. Tässä tapauksessa www.oppari.com, testi.oppari.com sekä 192.168.0.115. Lisäksi on tärkeää huomata, että itse CN tulee löytyä alternate_names-kentästä. Tämä sen vuoksi että sertifikaatin tarkistus tehdään ainoastaan SAN-kentästä, mikäli sinne on määritelty vaihtoehtoisia nimiä.

Tämän jälkeen tehdään pyyntö allekirjoittaa itse sertifikaatti komennolla:

openssl ca -config intermediate/openssl.cnf \

-extensions server_cert -days 375 -notext -md sha256 \

-in intermediate/csr/www.oppari.com.csr.pem \

-out intermediate/certs/www.oppari.com.cert.pem

Kuviossa 29 nähdään kuinka komento muodostaa pyynnön allekirjoittamista varten sekä näyttää tiedot ennen hyväksymistä.

```
Using configuration from intermediate/openssl.cnf
Enter pass phrase for /root/ca/intermediate/private/intermediate.key.pem:
Check that the request matches the signature
Signature ok
Certificate Details:
        Serial Number: 4101 (0x1005)
        Validity
            Not Before: May 21 18:59:41 2017 GMT
            Not After : May 31 18:59:41 2018 GMT
        Subject:
            countryName               = FI
            stateOrProvinceName       = UUSIMAA
            localityName              = ESPOO
            organizationName          = OPPARI
            commonName                = www.oppari.com
        X509v3 extensions:
            X509v3 Basic Constraints:
                CA:FALSE
            Netscape Cert Type:
                SSL Server
            Netscape Comment:
                OpenSSL Generated Server Certificate
            X509v3 Subject Key Identifier:
                0A:3D:AC:35:E6:0D:2D:7D:54:EB:1D:73:A5:F9:69:83:9D:61:75:4B
            X509v3 Authority Key Identifier:
                keyid:82:AD:88:B3:25:F5:6A:DE:DF:0A:10:8B:FE:3A:53:EA:BC:47:05:FD
                DirName:/C=FI/ST=UUSIMAA/L=ESPOO/O=OPPARI/CN=oppari.com
                serial:10:00

            X509v3 Key Usage: critical
                Digital Signature, Key Encipherment
            X509v3 Extended Key Usage:
                TLS Web Server Authentication
            X509v3 Subject Alternative Name:
                DNS:www.oppari.com, DNS:testi.oppari.com, IP Address:192.168.0.115
Certificate is to be certified until May 31 18:59:41 2018 GMT (375 days)
Sign the certificate? [y/n]:y


1 out of 1 certificate requests certified, commit? [y/n]y
Write out database with 1 new entries
Data Base Updated
```

Kuvio 29. Palvelimen sertifikaatin allekirjoituspyyntö

Tuloksena on allekirjoitettu sertifikaatti palvelimelle. Sertifikaattiketjun oikeudellisuuden voi tarkastaa muodostamalla ensiksi komennolla sertifikaatti, jossa on koko kyseinen ketju liitettynä:

cat intermediate/certs/intermediate.cert.pem \

certs/ca.cert.pem > intermediate/certs/ca-chain.cert.pem

Tätä ketjua vasten voidaan tarkastaa tehty palvelimen sertifikaatti komennolla:

openssl verify -CAfile /root/ca/intermediate/certs/ca-chain.cert.pem \ www.oppari.com.cert.pem

Komento antaa tulosteena kuvion 30 mukaisen sanoman mikäli sertifikaattiketju on luotu oikein.

```
root@ubuntu:~/ca/intermediate/certs# openssl verify -CAfile /root/ca/intermediate/certs/ca-chain.cert.pem \
> www.oppari.com.cert.pem
www.oppari.com.cert.pem: OK
```

Kuvio 30. Sertifikaattiketjun tarkistus

8.6 Apache2 SSL:n käyttöönotto

Sertifikaattien luonnin jälkeen voidaan web-palvelimelle konfiguroida HTPS-salaus käyttöön. Web-palvelimelle tuodaan keskitason sertifikaatti ja palvelimen oma sertifikaatti sekä yksityinen avain. Sertifikaatit sekä avain siirretään apache2 käyttämään ssl-kansioon, joka löytyy /etc/apache2/ssl/.

Web-palvelimelle konfiguroidaan käyttöön https muokkaamalla tiedostopolussa /ect/apache2/sites-available/default-ssl.conf olevaa tiedostoa. Default-ssl.conf-tiedosto muutetaan niin että se käyttää juuri siirrettyjä sertifikaatteja sekä yksityistä avainta. Konfiguraatio-tiedosto näyttää muokkauksen jälkeen kuvion 31 mukaiselta.

```
<IfModule mod_ssl.c>
    <VirtualHost _default_:443>
        ServerAdmin admin@example.com
        ServerName your_domain.com
        ServerAlias www.your_domain.com
        DocumentRoot /var/www/oppari.com/public_html
        ErrorLog ${APACHE_LOG_DIR}/error.log
        CustomLog ${APACHE_LOG_DIR}/access.log combined
        SSLEngine on
        SSLCertificateFile /etc/apache2/ssl/www.oppari.com.cert.pem
        SSLCertificateKeyFile /etc/apache2/ssl/www.oppari.com.key.pem
        SSLCertificateChainFile /etc/apache2/ssl/intermediate.cert.pem
        <FilesMatch "\.(cgi|shtml|phtml|php)$">
                        SSLOptions +StdEnvVars
        </FilesMatch>
        <Directory /usr/lib/cgi-bin>
                        SSLOptions +StdEnvVars
        </Directory>
        BrowserMatch "MSIE [2-6]" \
                        nokeepalive ssl-unclean-shutdown \
                        downgrade-1.0 force-response-1.0
        BrowserMatch "MSIE [17-9]" ssl-unclean-shutdown
    </VirtualHost>
</IfModule>
```

Kuvio 31. Apache2 default-ssl.conf -tiedosto

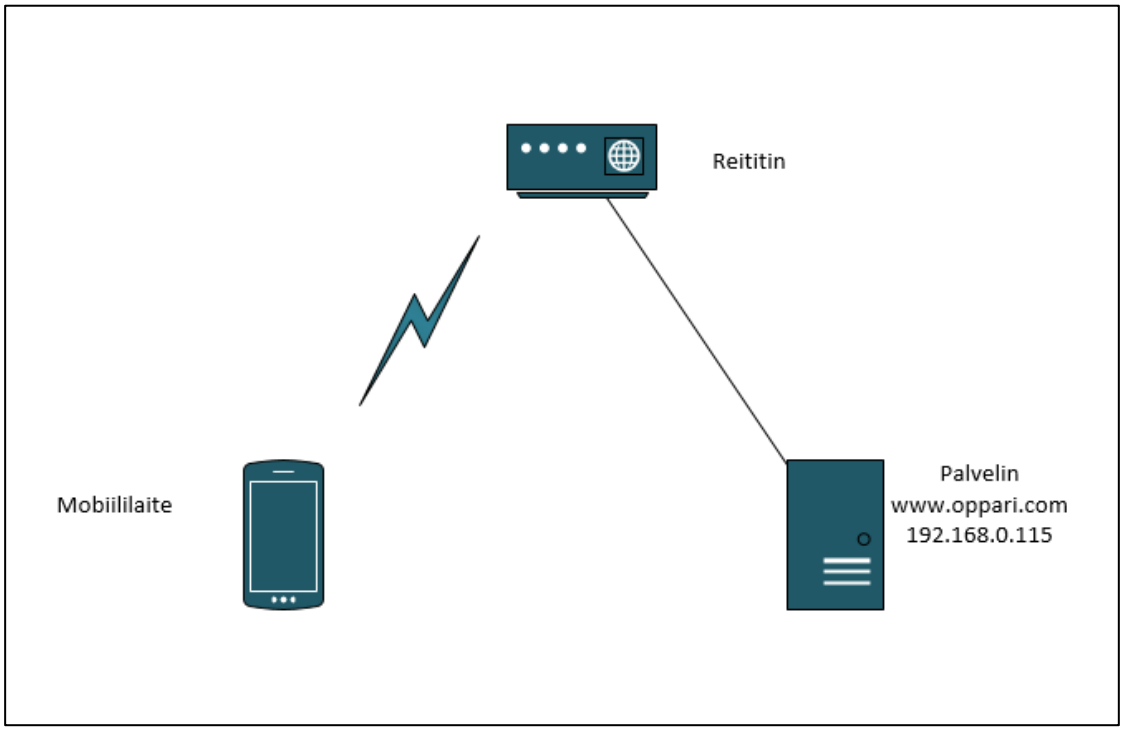
Seuraavilla komennoilla otetaan käyttöön tehdyt muutokset ja käynnistetään itse web-palvelu.

sudo a2ensite default-ssl.conf

sudo service apache2 restart

9 Testaus

Ympäristön pystytyksen jälkeen testattiin salauksen toiminta mobiililaitteella, emulaattorilla sekä selaimella yhdistäessä. Kuviossa 32 on havainnollistettu selkeämmin tilannetta testauksen suhteen.



Kuvio 32. Testauslaitteisto

Palvelin on asetettu IP-osoitteeseen 192.168.0.115 sekä sille on anettu FQDN www.oppari.com.

9.1 Mobiililaitteella testaaminen

Mobiililaitteelle asennettiin aikaisemmin tehty ca.cert.pem sertifikaatti, jonka avulla voidaan luottaa suoraan palvelimelta tulevaan sertifikaattiin. Tämän tyyppinen ratkaisu on esimerkki siitä, kun hankitaan luotettavalta taholta sertifikaattivarmenne palvelimelle. Tällöin mobiililaiteluottaa sertifikaatin myöntäjään automaattisesti ilman virheilmoituksia.

Cordovaan täytyy ensiksi määritellä alustat, joille halutaan ajaa sovellusta. Tämä tapahtuu cordova-projektin sisällä komennolla:

cordova platform add android

Tulosteena komennolle tulee suoritettu Android alusta lisäys kuten kuviossa 33 on nähtävillä.

```
toor@toor:~/oppari$ cordova platform add android
Adding android project...
Creating Cordova project for the Android platform:
        Path: platforms/android
        Package: com.oppari
        Name: oppari
        Activity: MainActivity
        Android target: android-25
Subproject Path: CordovaLib
Android project created with cordova-android@6.1.2
Installing "cordova-plugin-inappbrowser" for android
```

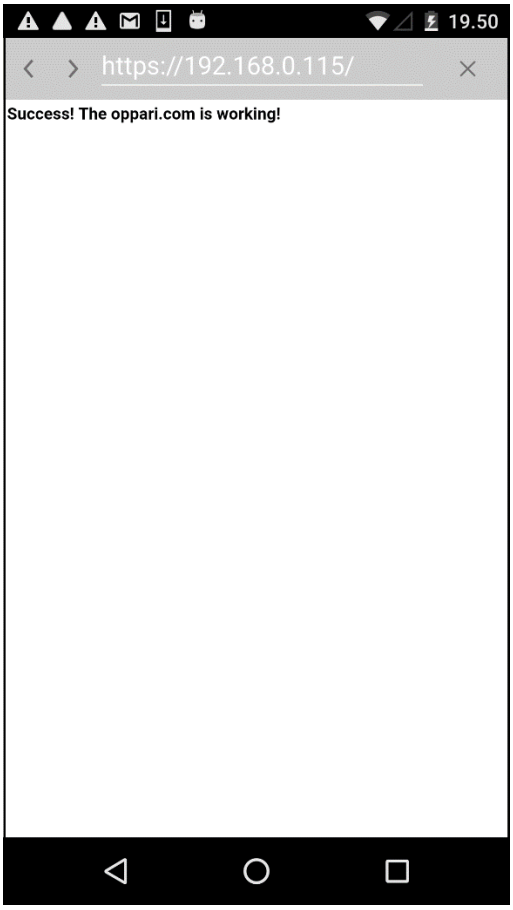
Kuvio 33. Android alustan lisääminen cordovan

Tämän jälkeen voidaan sovellus rakentaa ja ajaa käynnistymään tietokoneeseen liitettyyn laitteeseen komennoilla:

cordova build android

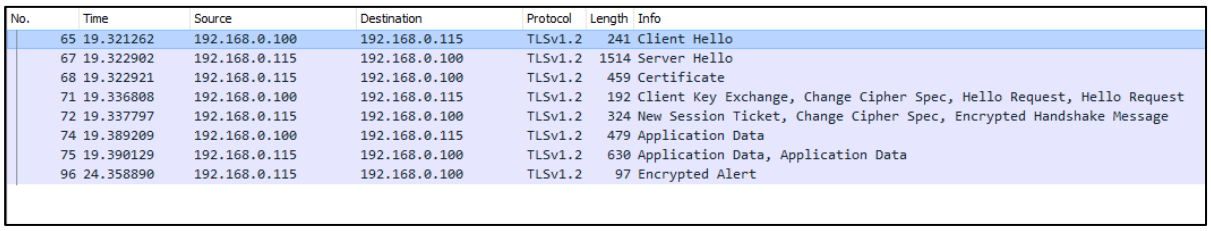
cordova run android

Kuviossa 34 on nähtävillä, kun applikaatio on käynnistetty puhelimeen ja se on yhdistetty web-palvelimelle onnistuneesti HTTPS-yhteydellä. Yhdistäminen tapahtuu palvelimen IP-osoitteeseen 192.168.0.115.



Kuvio 34. HTTPS-yhteyden muodostaminen mobiililaitteesta

Liikenteen tutkimiseksi kaapattiin myös Wiresharkin avulla palvelimen ja mobiililaitteen välinen kättely. Kuviosta 35 voidaan todentaa salauksen muodostuneen onnistuneesti.



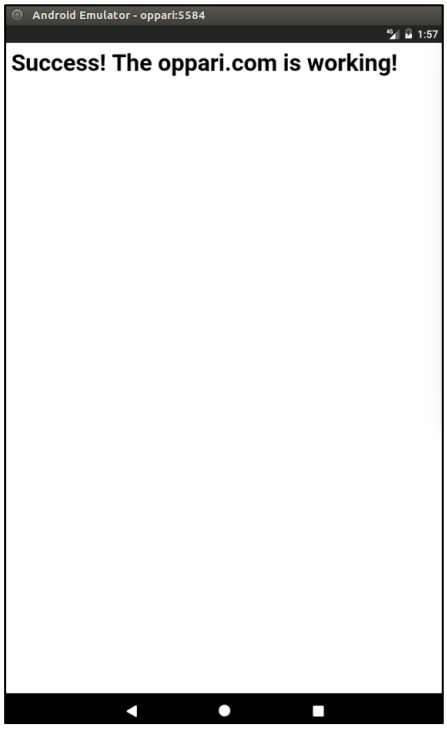
No.	Time	Source	Destination	Protocol	Length	Info
65	19.321262	192.168.0.100	192.168.0.115	TLSv1.2	241	Client Hello
67	19.322902	192.168.0.115	192.168.0.100	TLSv1.2	1514	Server Hello
68	19.322921	192.168.0.115	192.168.0.100	TLSv1.2	459	Certificate
71	19.336808	192.168.0.100	192.168.0.115	TLSv1.2	192	Client Key Exchange, Change Cipher Spec, Hello Request, Hello Request
72	19.337797	192.168.0.115	192.168.0.100	TLSv1.2	324	New Session Ticket, Change Cipher Spec, Encrypted Handshake Message
74	19.389209	192.168.0.100	192.168.0.115	TLSv1.2	479	Application Data
75	19.390129	192.168.0.115	192.168.0.100	TLSv1.2	630	Application Data, Application Data
96	24.358890	192.168.0.115	192.168.0.100	TLSv1.2	97	Encrypted Alert

Kuvio 35. Wireshark kaappaus liikenteestä

Tämän lisäksi salausta testattiin emulaattorilla ilman InAppBrowser-pluginia. Emulaattori käynnistettiin komennolla:

cordova emulate android

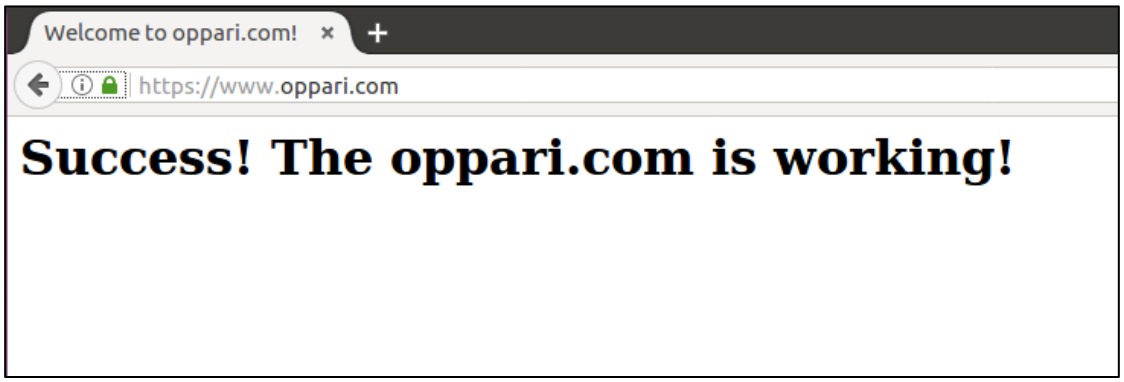
Tämän jälkeen emulaattori käynnistyy ja avaa applikaation. Kuviossa 36 on nähtävissä, kun emulaattori on yhdistynyt verkkosivulle.



Kuvio 36. Salauksen muodostuminen emulaattorilla

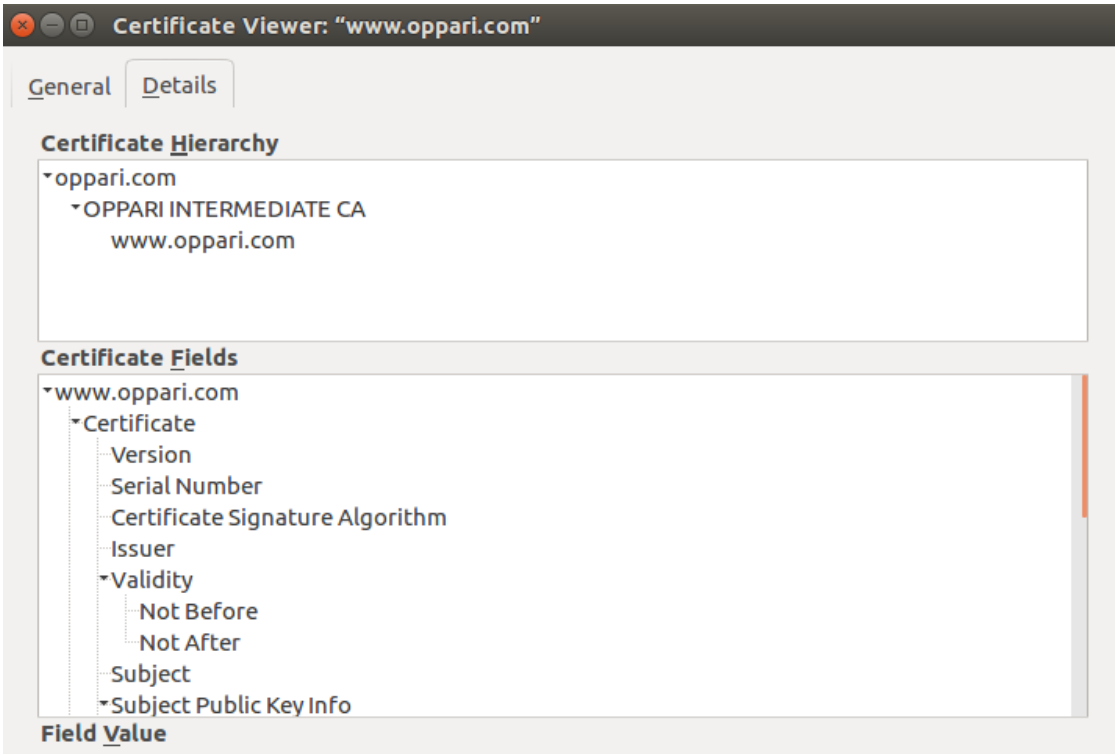
9.2 Selaimella testaaminen

Julkiseen osoitteeseen yhdistäminen voidaan todentaa yhdistämällä muun muassa Linux-tietokoneelta selaimelle osoitteeseen www.oppari.com. Kuviossa 37 nähdään onnistunut yhteys sekä salauksen tunnistukseksi vihreä lukon kuva.



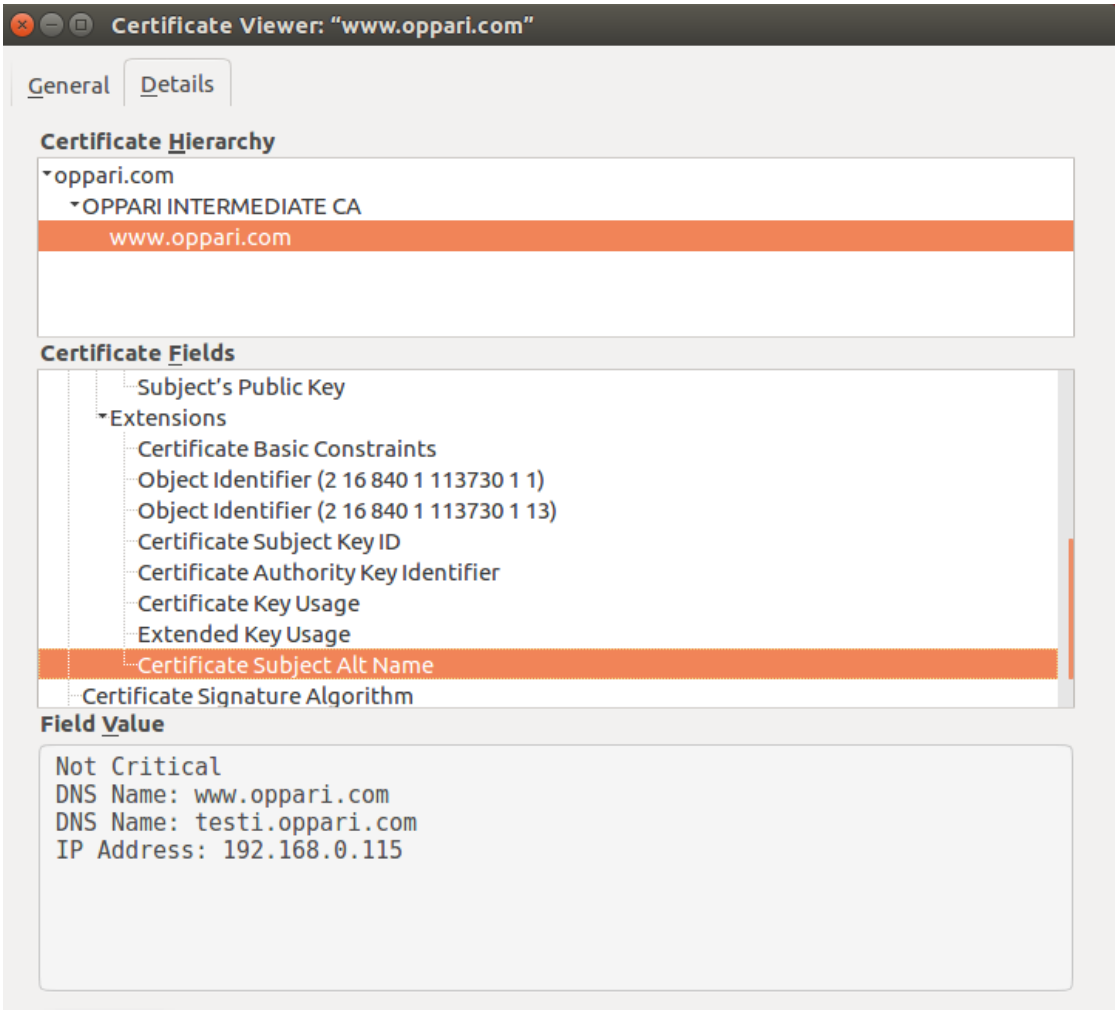
Kuvio 37. Salauksen muodostuminen selaimella

Sertifikaattia päästään tarkastelemaan tarkemmin klikkaamalla lukkoa. Kuviossa 38 tarkastellaan sertifikaatin hierarkiaa. Hierarkia muodostuu suoraan www.oppari.com jälkeen keskitason sertifikaattiin, joka on allekirjoittanut www.oppari.com-sertifikaatin. Ylimpänä hierarkiassa näkyy juurisertifikaatti oppari.com



Kuvio 38. Sertifikaatin hierarkkia

Tämän lisäksi kuviossa 39 on nähtävissä sertifikaatin vaihtoehtoiset nimet, jotka olivat www.oppari.com, testi.oppari.com ja 192.168.0.115.



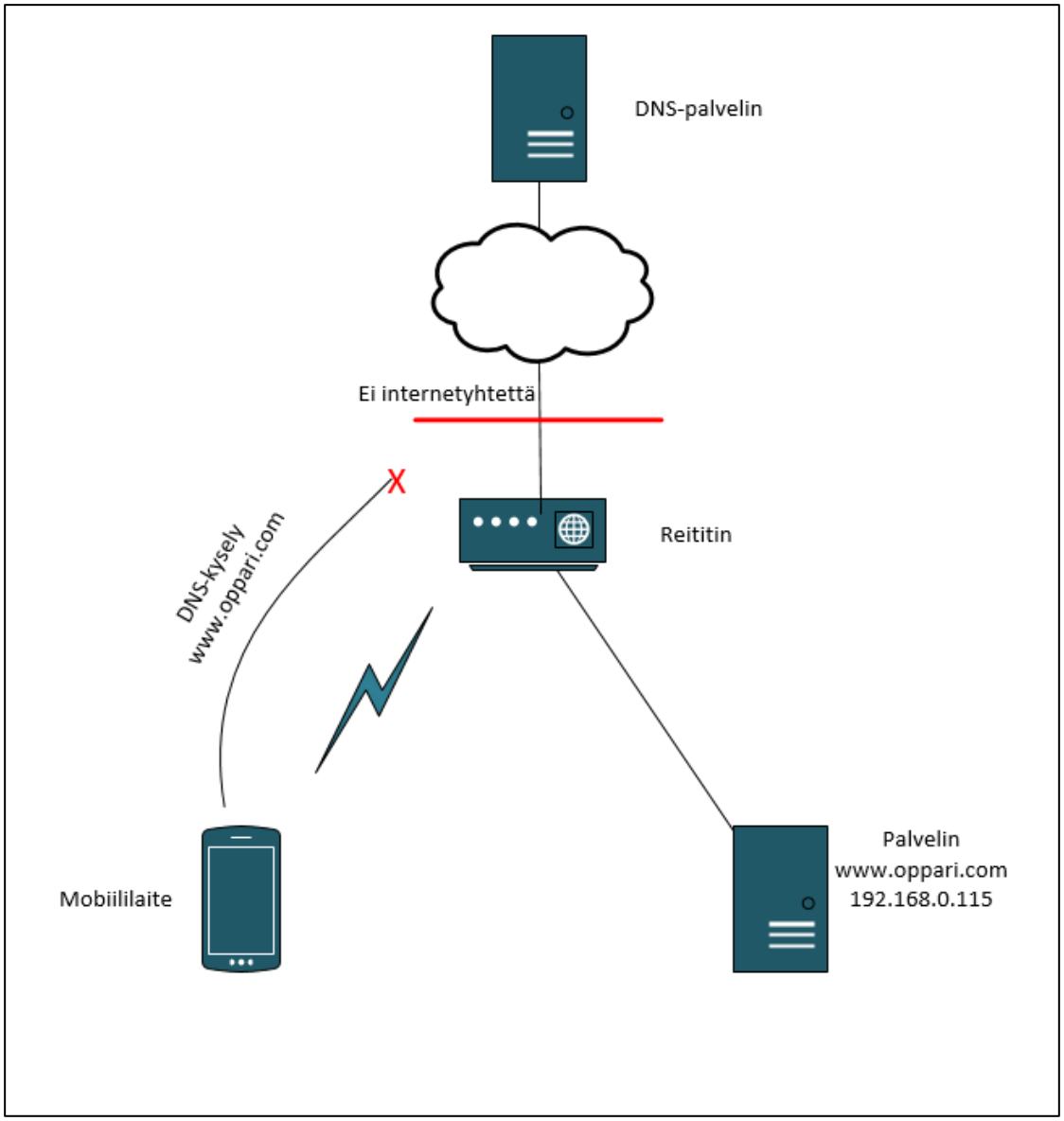
Kuvio 39. Sertifikaatin vaihtoehtoiset nimet

9.3 Havaitut ongelmat

Vaikka testiympäristössä salaus havaittiin onnistuneeksi sekä julkisen että yksityisen IP-osoitteen kautta on tämänlainen ratkaisu mahdoton julkisesti luotetuilla sertifikaateilla. Kaikki julkisesti luotetut CA:t kieltäytyvät allekirjoittamasta sertifikaatteja, joissa on joko yksityisverkon IP-osoite (esimerkiksi 10.0.0.0, 172.16.0.0 tai 192.168.0.0) tai paikallinen palvelinosoite. Tämä koskee myös multi-domain sertifikaatteja sekä wildcard-serfitikaatteja. Lisäksi subject alternate -vaihtoehdot ovat pois rajatut. (Beattie 2016)

Palvelimelle on mahdollista asentaa wildcard-sertifikaatti, jonka avulla voitaisiin ottaa käyttöön monia alidomain nimiä. Tässäkin piilee omat ongelmansa. Julkisen

osoitteeseen tapahtuvaa DNS-käännöstä ei voida suorittaa kotiverkossa, mikäli internetyhteys on katkennut ulkoverkkoon. Tämän työn yksi lähtökohdista oli salauksen toimiminen ilman internetyhteyttä. On lisäksi erittäin vaikeaa saada muutettua mobiililaitteisiin minkäänlaisia ohjauksia IP-osoitteiden ja domainnimen välille, ellei liki mahdotonta. Kuviossa 40 on havainnollistettu ongelmaa DNS-kyselyn kanssa.



Kuvio 40. DNS-kysely ilman internetyhteyttä

On mahdollista saada luotetuilta sertifikaatin myöntäjiltä yksityiskäyttöön tarkoitettuja sertifikaatteja, joissa on varattuja IP-osoitteita tai ei julkisia domainnimiä. Kuitenkin nämä sertifikaatit allekirjoittavat niin kutsutut ”private rootit”, jotka taas eivät

ole luotettu automaattisesti millään laitteella. Toisin sanoen samaan lopputulokseen päädyttäisiin käyttämällä itse allekirjoitettuja sertifikaatteja.

Cordovan mobiiliympäristön kuitenkin pitäisi mahdollistaa se, että HTTPS-yhteys voidaan tehdä itse allekirjoitetuilla sertifikaateilla. Tämä perustuu käytettyyn InAppbrowser-pluginiin. Sen avulla voidaan muodostaa valmiiksi luotettu yhteys ilman minkäänlaista ilmoitusta sertifikaatista, joka ei ole luotettu. Tämä toteutustapa vaatii juurisertifikaatin asentamista itse mobiilisovelluksen sisään ja lukemista sieltä, kun sovellus käynnistetään ja yhdistetään palvelimeen.

10 Yhteenveto

10.1 Pohdinta

Tavoitteena opinnäytetyössä oli tutkia mahdollisia toteutuksia mobiililaitteen ja palvelimen väliselle salaukselle lähiverkossa ilman internetyhteyttä. Mahdollisia eri vaihtoehtoja tutkimuksessa oli paljon mutta lopulta yksinkertainen ja helppo toteutustapa osoittautui parhaaksi. Tavoitteena oli tutkia mahdollisia toteutustapoja ja siinä mielessä tavoite saavutettiin. Työn tuloksena kuitenkin joudutaan toteamaan, ettei saatu aikaiseksi järkevää mallia muodostaa salausta ilman ilmoitusta epäluotetuista sertifikaateista. Kuitenkin lisäkehityksellä ajan kanssa on mahdollista saada toimiva ratkaisumalli ongelmaan.

Opinnäytetyö oli melko haastava ja lähtökohdat olivat vaikeat. Ongelmia tuli vastaan paljon sekä paljon tuli uuden opettelemista, jotka tekivät työstä mielenkiintoisen. Sertifikaattien kanssa työskentelystä oli ennestään kokemusta mutta itse mobiiliapplikaatioista ei juurikaan. Linux-käyttöjärjestelmästä sekä VMware-ympäristöstä oli kokemusta sekä työskentely virtualisoinnin kanssa oli helppoa. Opinnäytetyö opetti kuitenkin paljon mobiiliapplikaatioista varsinkin Cordovan puolella sekä sertifikaattien kanssa toimimisesta.

Yllättävää oli se, kuinka hyvin kaikki järjestelmät toimivat yhteen virtualisoinnista huolimatta. Ei tullut juurikaan eteen ongelmia yhteensopivuuksien tai yhteysongelmien kanssa. Paljon tuli luettua itselle uutta muun muassa mobiiliapplikaatioiden puolelta. Opinnäytetyö kasvatti myös kiinnostusta sovelluskehittämisen puolelle.

10.2 Jatkokehitysehdotukset

Ongelman ollessa ratkaistavissa olisi mielenkiintoista nähdä kuinka sertifikaatti saataisiin luettua mobiiliapplikaatiosta luotettavasti. Tämän ratkaisun jälkeen helpointa sertifikaattien kanssa olisi pystyttää oma CA-ympäristö. Tämä mahdollistaisi helpon sertifikaattien jakelun sekä sertifikaatteihin voitaisiin itse vaikuttaa paremmin. Sertifikaattien kanssa tulee toimia varovaisesti ja suojata ne mahdollisilta väärinkäytöiltä. Tarpeeksi vahvan kryptauksen käyttäminen avainten kanssa on suositeltavaa. CA-ympäristössä tulisi olla mahdollisuus tehdä revokaatio sertifikaateille, mikäli yksityinen avain paljastuu. Mikäli käyttöön tulee CA-ympäristö ja itse allekirjoitetut sertifikaatit olisi hyvä ottaa käyttöön sertifikaatin kiinnittäminen. Se lisäisi tietoturvaa, sillä palvelimen olisi tarjottava oikeanlaista sertifikaattia, jotta yhteys voitaisiin muodostaa. Kiinnittäminen estäisi mahdolliset MITM-hyökkäykset tehokkaasti.

Tämän lisäksi kehityksen kohteena on palvelimen asema lähiverkossa. Opinnäytetyössä lähtökohtana oli, että palvelimella on olemassa kiinteä IP-osoite, jonka pohjalta voidaan sertifikaatti tehdä lähiverkkoon. Ongelmaksi muodostuu se, ettei lähiverkoissa yleensä ole kiinteän IP-osoitteen mahdollisuutta, ilman sellaisen konfiguroimista käsin ja tietämällä etukäteen ympäristöstä, jottei tule päällekkäisyyksiä. Kotiverkossa kun yleensä käytetään DHCP:tä jakamaan IP-osoitteet laitteille.

Lähteet

Alakoski, T. 2003. Avainten vaihto ja jakelu IPSec-järjestelmässä. Viitattu 12.3.2017. http://www.cs.tut.fi/tlt/npg/icefin/documents/Teemun-dippa.pdf

ARP poisoning attack and mitigation techniques. 2016. http://www.cisco.com/c/en/us/products/collateral/switches/catalyst-6500-series-switches/white_paper_c11_603839.html

Barret, D., Byrnes, G. & Silverman, R. 2001. SSH the secure shell. Viitattu 20.3.2017. https://courseweb.pitt.edu/bbcswebdav/institution/Pitt%20Online/MLIS_Pitt_Online/LIS%202600/Intro%20Module/SSH_Second_Edition.pdf

Certificate pinning. 2016. Artikkeli Bluecoatin kotisivuilta. Viitattu 24.4.2017. https://www.bluecoat.com/ko/documents/download/7ff09c94-7b88-4319-a766-191c9dedde22

Cobb, M. 2016. Transport Layer Security. Viitattu 24.2.2017. http://searchsecurity.techtarget.com/definition/Transport-Layer-Security-TLS

Cozify Oy. 2017. Cozify Oy:n kotisivut. Viitattu 17.3.2017 http://www.cozify.fi/

Finley, K. 2014. It's time to encrypt the entire internet. Viitattu 19.3.2017. https://www.wired.com/2014/04/https/

Fisher, D. 2013. What is a man-in-the-middle attack? Viitattu 27.3.2017. https://blog.kaspersky.com/man-in-the-middle-attack/1613/

Fisher, T. 2016. What is a checksum? Viitattu 9.4.2017. https://www.lifewire.com/what-does-checksum-mean-2625825

Beattie, D. 2016. SSL/TLS certificates for internal servers. https://www.globalsign.com/en/blog/certificates-for-internal-servers/

Hassinen, A., Malinen, M. & Miettinen I. 2001. Miten virheitä korjataan. Viitattu 9.4.2017. https://www.netlab.tkk.fi/opetus/s38118/s00/tyot/8/virhe.shtml

Johansen, B. 2015. HTTP public key pinning (HPKP). Viitattu 22.4.2017. https://www.bjornjohansen.no/public-key-pinning

Kangas, E. 2016. SSL versus TLS -What's the difference? Viitattu 24.4.2017. https://luxsci.com/blog/ssl-versus-tls-whats-the-difference.html

Lim, N. 2016. What is SSL? A beginner's guide to SSL encryption. Viitattu 19.3.2017. https://blog.webnames.ca/what-is-ssl/

Man in the Middle (MITM) Attack. Artikkeli incapsula www-sivulla. N.d. Viitattu 27.3.2017. https://www.incapsula.com/web-application-security/man-in-the-middle-mitm.html

Miessler, D. 2005. Security: identification, authentication and authorization. Viitattu 2.4.2017.

https://danielmiessler.com/blog/security-identification-authentication-and-authorization/

Nikkonen, V. 2010. Tietoturvalliset etäyhteydet IPSec VPN-tekniikan avulla. Viitattu 12.3.2017. https://www.theseus.fi/bitstream/handle/10024/10802/Ville%20Nikkonen_402T04.pdf?sequence=1

Olenski, J. 2016. SSL vs. TLS – what's the difference? https://www.globalsign.com/en/blog/ssl-vs-tls-difference/

Rouse, M. 2015. Knowledge factor. Viitattu 2.4.2017. http://searchsecurity.techtarget.com/definition/knowledge-factor

Rouse, M. 2016. Secure Shell (SSH). Viitattu 12.3.2017. http://searchsecurity.techtarget.com/definition/Secure-Shell

Shinder, D. 2005. Solutionbase: Introduction to SSL VPNs. Viitattu 15.3.2017. http://www.techrepublic.com/article/solutionbase-introduction-to-ssl-vpns/

SSL how it works. 2009. Awardspace www-sivut. Viitattu 19.3.2017. https://www.awardspace.com/wp-content/uploads/2009/10/img_ssl_how_it_works_1.jpg

The relentless growth of cybercrime. 2016. Artikkeli europolin kotisivuilla. Viitattu 19.3.2017. https://www.europol.europa.eu/newsroom/news/relentless-growth-of-cybercrime

Vaughan, M. 2014. Why your site should be using HTTPS. Viitattu 19.3.2017. https://www.chapterthree.com/blog/why-your-site-should-be-using-https

What are knowledge factors, possession factors and inherence factors? 2016. Blogi proofid yrityksen www-sivuilla. Viitattu 2.4.2017. https://www.proofid.co.uk/blog/knowledge-factors-possession-factors-inherence-factors/

What is DNS spoofing? 2017. Artikkeli keycdn www-sivuilla. Viitattu 24.4.2017. https://www.keycdn.com/support/dns-spoofing/

What is hypertext transfer protocol secure? Artikkeli brickmarketing www-sivuilla. N.d. Viitattu 24.2.2017. http://www.brickmarketing.com/define-hypertext-transfer-protocol-secure.htm

What is multi-factor authentication? Blogi authanvil www-sivuilta. N.d. Viitattu 2.4.2017. https://authanvil.com/blog/what-is-multi-factor-authentication

What is SSL. 2016. Artikkeli Comodo yrityksen www-sivuilta. Viitattu 24.2.2017. https://ssl.comodo.com/ssl.php

Liitteet

Liite 1. Openssl-konfiguraatiotiedosto juurisertifikaattiin

```
[ ca ]
default_ca  = CA_default
[ CA_default ]
dir             = /root/ca
certs           = $dir/certs
crl_dir         = $dir/crl
database        = $dir/index.txt
new_certs_dir   = $dir/newcerts
certificate     = $dir/certs/cacert.pem
serial          = $dir/serial
crlnumber       = $dir/crlnumber
crl             = $dir/crl/ca.crl.pem
private_key     = $dir/private/ca.key.pem
RANDFILE        = $dir/private/.rand

name_opt        = ca_default
cert_opt        = ca_default

crl_extensions  = crl_ext
default_days    = 375
default_crl_days= 30
default_md      = sha256
preserve        = no
policy      = policy_strict

[ policy_strict ]
countryName             = match
stateOrProvinceName     = match
organizationName        = match
organizationalUnitName  = optional
commonName              = supplied
emailAddress            = optional

[ policy_loose ]
countryName             = optional
stateOrProvinceName     = optional
localityName            = optional
organizationName        = optional
organizationalUnitName  = optional
commonName              = supplied
emailAddress            = optional

[ req ]
default_bits        = 2048
distinguished_name  = req_distinguished_name
string_mask         = utf8only
x509_extensions     = v3_ca

[ req_distinguished_name ]
countryName                     = Country Name (2 letter code)
stateOrProvinceName             = State or Province Name (full name)
localityName                    = Locality Name (eg, city)
0.organizationName              = Organization Name (eg, company)
organizationalUnitName          = Organizational Unit Name (eg, section)
commonName                      = Common Name
emailAddress                    = Email Address

[ v3_ca]
basicConstraints = CA:true
subjectKeyIdentifier=hash
authorityKeyIdentifier=keyid:always,issuer
keyUsage = critical, digitalSignature, cRLSign, keyCertSign

[ v3_intermediate_ca ]
subjectKeyIdentifier = hash
authorityKeyIdentifier = keyid:always,issuer
basicConstraints = critical, CA:true, pathlen:0
keyUsage = critical, digitalSignature, cRLSign, keyCertSign

[ server_cert ]
basicConstraints = CA:FALSE
nsCertType = server
nsComment = "OpenSSL Generated Server Certificate"
subjectKeyIdentifier = hash
authorityKeyIdentifier = keyid,issuer:always
keyUsage = critical, digitalSignature, keyEncipherment
extendedKeyUsage = serverAuth

[ crl_ext ]
authorityKeyIdentifier=keyid:always
```

Liite 2. Openssl-konfiguraatiotiedosto keskitason ja palvelimen sertifikaattiin

```
[ ca ]
default_ca  = CA_default
[ CA_default ]
dir             = /root/ca/intermediate
certs           = $dir/certs
crl_dir         = $dir/crl
database        = $dir/index.txt
new_certs_dir   = $dir/newcerts
certificate     = $dir/certs/intermediate.cert.pem
serial          = $dir/serial
crlnumber       = $dir/crlnumber
crl             = $dir/crl/ca.crl.pem
private_key     = $dir/private/intermediate.key.pem
RANDFILE        = $dir/private/.rand

name_opt        = ca_default
cert_opt        = ca_default

crl_extensions  = crl_ext
default_days    = 375
default_crl_days= 30
default_md      = sha256
preserve        = no
policy      = policy_loose

[ policy_strict ]
countryName             = match
stateOrProvinceName     = match
organizationName        = match
organizationalUnitName  = optional
commonName              = supplied
emailAddress            = optional

[ policy_loose ]
countryName             = optional
stateOrProvinceName     = optional
localityName            = optional
organizationName        = optional
organizationalUnitName  = optional
commonName              = supplied
emailAddress            = optional

[ req ]
default_bits        = 2048
distinguished_name  = req_distinguished_name
string_mask         = utf8only
x509_extensions     = v3_ca

[ req_distinguished_name ]
countryName                     = Country Name (2 letter code)
stateOrProvinceName             = State or Province Name (full name)
localityName                    = Locality Name (eg, city)
0.organizationName              = Organization Name (eg, company)
organizationalUnitName          = Organizational Unit Name (eg, section)
commonName                      = Common Name
emailAddress                    = Email Address

[ v3_ca]
basicConstraints = CA:true
subjectKeyIdentifier=hash
authorityKeyIdentifier=keyid:always,issuer
keyUsage = critical, digitalSignature, cRLSign, keyCertSign

[ v3_intermediate_ca ]
subjectKeyIdentifier = hash
authorityKeyIdentifier = keyid:always,issuer
basicConstraints = critical, CA:true, pathlen:0
keyUsage = critical, digitalSignature, cRLSign, keyCertSign

[ server_cert ]
basicConstraints = CA:FALSE
nsCertType = server
nsComment = "OpenSSL Generated Server Certificate"
subjectKeyIdentifier = hash
authorityKeyIdentifier = keyid,issuer:always
keyUsage = critical, digitalSignature, keyEncipherment
extendedKeyUsage = serverAuth
subjectAltName = @alternate_names

[ alternate_names ]

DNS.1   = www.oppari.com
DNS.2   = testi.oppari.com
IP.1    = 192.168.0.115

[ crl_ext ]
authorityKeyIdentifier=keyid:always
```